# Learning from Mistakes - A Framework for Neural Architecture Search

**Bhanu Garg[1*], Li Zhang[2*], Pradyumna Sridhara[1], Ramtin Hosseini[1],**
**Eric Xing[3], Pengtao Xie[1]**

[1] University of California, San Diego, USA
[2] Zhejiang University, China
[3] Carnegie Mellon University, USA; Mohamed bin Zayed University of Artificial Intelligence, UAE

## Abstract

Learning from one's mistakes is an effective human learning technique where the learners focus more on the topics where mistakes were made, so as to deepen their understanding. In this paper, we investigate if this human learning strategy can be applied in machine learning. We propose a novel machine learning method called Learning From Mistakes (LFM), wherein the learner improves its ability to learn by focusing more on the mistakes during revision. We formulate LFM as a three-stage optimization problem: 1) learner learns; 2) learner re-learns focusing on the mistakes, and; 3) learner validates its learning. We develop an efficient algorithm to solve the LFM problem. We apply the LFM framework to neural architecture search on CIFAR-10, CIFAR-100, and Imagenet. Experimental results strongly demonstrate the effectiveness of our model.

## Introduction

Over the years, humans have accumulated a lot of practical learning techniques. One such effective learning method is to learn from previous mistakes. Initially, the learner learns a concept and evaluates themselves through a test to measure their level of understanding. The topics in the concept where the learner makes more mistakes can be identified as not having been learned well by the learner. Therefore, the learner will re-study the topic while focusing on the topics where mistakes were made. This will ensure that repetition of similar mistakes in the future is avoided while also strengthening previously well-learned topics. Inspired by this human learning technique, we propose a methodology that can apply the idea of learning from mistakes to machine learning.

With the deep learning era kicking off in machine learning, state-of-the-art neural network performance is achieved mainly by architectures designed manually by experts. The neural architecture search (NAS) and evaluation by human experts require substantial effort and may not give the most optimal performance. Recently, there has been growing interest in automating the manual process of learning architecture design. This paper proposes a general framework that draws inspiration from human learning skills and can be applied to any differentiable architecture search algorithm. We also explore the efficacy of our learning method on NAS.

In our framework, the model consists of two sets of network weights sharing a common learnable architecture, a learnable data encoder, and a coefficient vector - representing different components of the learner. The two sets of network weights correspond to two parts of the main task learning faculties of the learner. The learnable architecture corresponds to skill learning faculty of the learner. The data encoder and coefficient vector correspond to auxiliary faculties of the learner, helping it to recognise and summarize the information respectively. We propose a multi-level optimization framework that uses the above sets of parameters to learn better neural architectures.

We begin by training the first set of network weights on a training dataset. We then see what mistakes our model makes while predicting on the validation set. Then, for each training example, we assign specific weights based on the mistakes made by the model and the similarity of this example to an incorrectly predicted validation example. The second set of network weights are trained on these weighted examples, essentially making the model learn from its mistakes and correct them. The vanilla approaches in NAS do not factor in the variations in learning difficulty. In contrast, our method re-weights the training examples at each stage based on the current capability of the learner, enabling the learner to deal with more challenging cases in the unseen data. Then finally, the architecture, encoder, and coefficient vectors are learned on the second model's validation performance.

The major contributions of the paper are as follows:

- Inspired by the human learning technique, we propose a novel approach to apply the Learning from Mistakes (LFM) method in machine learning.

- We formulate LFM as a multi-level optimization framework that involves three stages of learning: learner learns; learner corrects its mistakes; learner, encoder, and coefficient vector validate themselves.

- We apply our approach to neural architecture search on CIFAR-100, CIFAR-10, and ImageNet. The results demonstrate the effectiveness of our method.

---

[*]These authors contributed equally.

## Related Works

### Data Re-weighting and Selection

Several approaches have been proposed for data selection. Matrix column subset selection (Deshpande and Rademacher 2010; Boutsidis, Mahoney, and Drineas 2009) aims to select a subset of data examples that can best reconstruct the entire dataset. Similarly, Coreset selection (Bachem, Lucic, and Krause 2017) chooses representative training examples such that models trained on the selected examples have comparable performance with those trained on all training examples. These methods perform data selection and model training separately. As a result, the validation performance of the model cannot be used to guide data selection. (Ren et al. 2018) proposes a meta-learning method to learn the weights of training examples by performing a meta gradient descent step on the weights of the current mini-batch of examples. (Shu et al. 2019) propose a method that can adaptively learn an explicit weighting function directly from data.

Our work takes inspiration from (Ren et al. 2018) to use a meta gradient step on the weights, and from (Shu et al. 2019) to explicitly learn a weighting function and extends them to use not only validation and training performance but also the similarity of data and labels. We make this possible by learning the weights and the learner in different stages of the optimization problem. Moreover, the previous works focus on selecting training (finetuning) examples using a bi-level optimization framework, while our work focuses on selecting pretraining examples using a three-level optimization framework.

### Neural Architecture Search (NAS)

Recently, NAS has come to the forefront of deep learning techniques due to its success in discovering neural architectures that can substantially outperform manually designed ones. Early versions of NAS such as (Zoph and Le 2017; Pham et al. 2018; Zoph et al. 2018) used computationally intensive approaches like reinforcement learning - where the accuracy of the validation set was defined as the reward, and a policy network was trained to generate architectures that can maximize these rewards. Another contemporary approach (Liu et al. 2018b; Real et al. 2019a) was using evolutionary learning techniques - where the set of all architectures represent a population and the fitness score is the validation accuracy of each architecture. Architectures with lower fitness scores would be replaced with higher fitness score architectures. However, even this approach was computationally intensive. To address this problem, differentiable architecture search techniques were explored (Cai, Zhu, and Han 2019; Liu, Simonyan, and Yang 2019; Xie et al. 2019) and their results are much more promising because of the use of weight-sharing techniques and the application of gradient descent in a continuous architecture search space.

DARTS (Liu, Simonyan, and Yang 2019) made the first breakthrough in the area of Differentiable NAS. Several other DARTS-based techniques (Chen et al. 2021; Xu et al. 2021; Liang et al. 2019a; Chu et al. 2021) have explored to further reduce the cost of computation for differentiable NAS. Some of the approaches include - PDARTS (Chen et al. 2021) increasing the depth of architectures progressively during the searching, PC-DARTS (Xu et al. 2021) evaluating only a sub-set of channels and thereby reducing the redundancy in the search space.

The LFM framework proposed in this paper can be applied to any differentiable NAS method for further enhancement.

## Method

In this section, we propose a framework that can imitate human learning in the form of Learning from Mistakes (LFM) and present an optimization algorithm for solving the problem of LFM.

### The Learning from Mistakes framework

The framework contains two sets of network weights $W_1$ and $W_2$ - that are two parts of the same learner and are trying to learn to perform the same target task. They share a learnable architecture $A$. The primary goal here is to help the learner correct the mistakes (made when studying for the first time) during the revision. Further, to help map the topics in the test to topics in the syllabus, there is an encoder with pre-defined neural architecture (by human experts) with learnable network weights $V$; and a coefficient vector $r$. We organized the learning into three stages.

**Stage I.** In the first stage, we train the first set of network weights $W_1$ by minimizing the loss on the training dataset $D^{(tr)}$. The optimal weights $W_1^*(A)$ is a function of architecture A, which at this stage is fixed, and hence:

$$W_1^*(A) = \arg\min_{W_1} L(A, W_1, D^{tr})$$

The architecture $A$ is used to define the training loss but is not updated at this stage. If we were to directly learn $A$ by minimizing this training loss, a trivial solution would be yielded where $A$ is very large and complex, and would perfectly overfit the training data but generalize poorly on unseen data.

**Stage II.** In the second stage, the goal is to re-weight the training samples for training the second set of network weights $W_2$ of the learner. We apply $W_1^*(A)$ to the validation dataset $D^{(val)}$ and check its performance on the validation examples. Without loss of generality, we assume the task is image classification. To make the model re-learn whilst focusing more on mistakes on the validation examples by $W_1^*(A)$, we re-weight each training example $d_i^{(tr)}$ based on the following metrics:

- Visual similarity between $d_i^{(tr)}$ and $d_j^{(val)}$, denoted by $x_{ij}$

- Label similarity of $d_i^{(tr)}$ and $d_j^{(val)}$, denoted by $z_{ij}$

- Valid performance of $W_1^*(A)$ on $D^{(val)}$, denoted by $u_j$

In human learning, a question that has been incorrectly learned can be corrected during the relearning stage by focusing more on examples that are similar to the wrongly

learned question. Here, the metric $x_{ij}$ tries to measure how similar a previously incorrectly predicted $d_j^{(val)}$ is to a training example $d_i^{(tr)}$ and $z_{ij}$ depicts whether they describe the same topic. The term $u_j$ measures by how much $W_1^*(A)$ is wrong for each validation example $j$.

We use these re-weighted training examples to train $W_2$. This allows $W_2$ to focus on the topics that $W_1$, after training, could not get right.

Visual similarity measures how similar the training example $d_i^{(tr)}$ is to each validation example $d_j^{(val)}$. Let V denote an image encoder. For each validation example $d_j^{(val)}$, its similarity with $d_i^{(tr)}$ is defined as the dot-product attention (Luong, Pham, and Manning 2015) as:

$$x_{ij} = \frac{exp(V(d_i^{(tr)}).V(d_j^{(val)}))}{\sum_{k=1}^{N^{(val)}} exp(V(d_i^{(tr)}).V(d_k^{(val)}))} \quad (1)$$

where $N^{(val)}$ is the number of validation examples. $V(d)$ denotes the $K$-dimensional visual representation of the data example d.

Label similarity measures the similarity between the label of the training example and the label of each validation example. Let $z_{ij}$ denote the label similarity between a validation example $d_j^{(val)}$ and a training example $d_i^{(tr)}$. We define $z_{ij}$ as:

$$z_{ij} = \mathbb{I}\{y_i^{(tr)} = y_j^{(val)}\} \quad (2)$$

where $y$ is the label of the corresponding data example, and $\mathbb{I}\{\}$ the indicator function on the condition being true or not.

The validation performance $u_j$ of $W_1^*(A)$ on a validation example $d_j^{(val)}$ is the cross entropy loss on this example:

$$u_j = \text{crossentropy}(f(d_j^{(val)}; W_1^*(A)), y_j^{(val)}) \quad (3)$$

where $f(d_j^{(val)}; W_1^*(A))$ is the predicted probabilities of $W_1^*(A)$ on $d_j^{(val)}$ and $y_j^{(val)}$ is the class label of $d_j^{(val)}$.

Let $x_i$, $z_i$, and $u$ be $N^{(val)}$-dimensional vectors where the $j$-th element is $x_{ij}$, $z_{ij}$, and $u_j$ defined before. We calculate the weight $a_i$ of the training example $d_i^{(tr)}$ as:

$$a_i = \text{sigmoid}((x_i \odot z_i \odot u)^T r) \quad (4)$$

where $\odot$ denotes element-wise multiplication, and $r$ is a co-efficient vector. Note $a_i$ is a function of $V$, $W_1^*(A)$, and $r$.

Given the weight $a_i$ of each training example, we train the second set of network weights $W_2$ by minimizing the weighted training loss, with the architecture $A$, encoder $V$, and $r$ fixed.

$$W_2^*(W_1^*(A), V, r) = \arg\min_{W_2} \sum_{i=1}^{N^{tr}} a_i \ell(A, W_2(A), d_i^{(tr)})$$
$$(5)$$

**Stage III.** In the third and final stage, the encoder $V$, coefficient vector $r$, and the architecture $A$ minimise the validation loss of $W_2^*(W_1^*(A), V, r)$.

$$A, V, r = \arg\min_{A,V,r} L(A, W_2^*(W_1^*(A), V, r), D^{(val)}) \quad (6)$$

Putting the above pieces together, we have the following LFM framework, which is a three level optimization problem:

$$A, V, r = \arg\min_{A,V,r} L(A, W_2^*(W_1^*(A), V, r), D^{(val)})$$
$$s.t. \quad W_2^*(A) = \arg\min_{W_2} \sum_{i=1}^{N^{tr}} a_i \ell(A, W_2(A), d_i^{(tr)}) \quad (7)$$
$$W_1^*(A) = \arg\min_{W_1} L(A, W_1, D^{tr})$$

We summarise the above equations in the process flow diagram Figure 1. In our end-to-end framework, $W_1$ will learn from its previous mistakes and avoid making similar mistakes, in an indirect way. After the weights of $W_2$ are trained by correcting the mistakes made by $W_1$, the architecture will be updated accordingly since the gradient of $A$ depends on $W_2$; an updated $A$ will render $W_1$ to change as well since the gradient of $W_1$ depends on $A$. Along the chain $W_2 \rightarrow A \rightarrow W_1$ chain, $W_1$ is indirectly influenced by $W_2$.
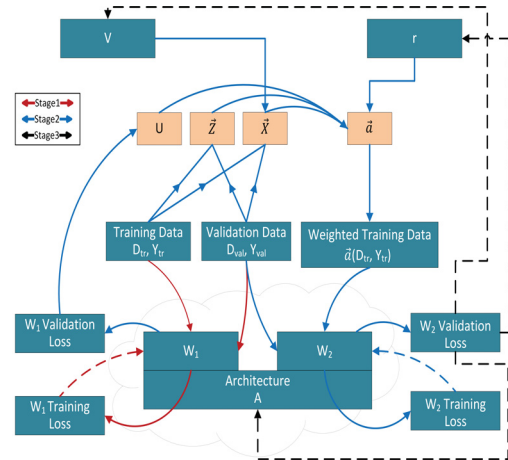


Figure 1: The overall process flow of our method when applying to NAS. The red arrows indicate stage 1 processes, blue arrows indicate stage 2 processes, and black arrows indicate stage 3 processes.

Similar to (Liu, Simonyan, and Yang 2019), we represent the architecture $A$ of the learner in a differentiable way. The search space of $A$ is composed of many building blocks, where the output of each block is associated with a weight $a$ indicating the importance of the block. After learning, block whose weight $a$ is among the largest are retained to form the final architecture. To this end, architecture search amounts to optimizing the set of architecture weights $A = \{a\}$. Our

framework can be applied on top of any differentiable architecture search methods such as DARTS (Liu, Simonyan, and Yang 2019), PDARTS (Chen et al. 2021), PC-DARTS (Xu et al. 2021), DARTS- (Chu et al. 2021) among others.

## Optimization Algorithm

In this section, we derive an optimization algorithm to solve the LFM problem defined in Equation 7. Inspired by (Liu, Simonyan, and Yang 2019), we approximate $W_1^*(A)$ and $W_2^*(W_1^*(A), A, V, r)$ one step gradient descent updates for the inner optimization equations to reduce the computational complexity.

**Stage I.** For Stage 1, we approximate $W_1^*(A)$ using one step descent for the loss on training data $L(A, W_1, D^{tr})$ keeping $A$ constant as follows:

$$W_1'(A) = W_1 - \eta_{W_1} \nabla_{W_1} L(A, W_1, D^{tr}) \qquad (8)$$

**Stage II.** For Stage 2, we use $W_1'$ from the Equation 8 to get $u_j$. We compute $x_i$ and $z_i$ for each training sample $d_i^{tr}$ using the equations 1 and 2, to finally compute $a_i$ as:

$$a_i(d^{val}, d_i^{tr}) = \text{sigmoid}((x_i \odot z_i \odot u)^T r) \qquad (9)$$

Next, we use one step gradient descent to approximate $W_2^*(A, W_1^*(A), V, r)$ as:

$$W_2'(A) = W_2 - \eta_{W_2} \nabla_{W_2} \sum_{i=1}^{N^{tr}} a_i \ell(A, W_2, d_i^{(tr)}) \qquad (10)$$

**Stage III.** For Stage 3, we plug Equation 10 to learn architecture $A$, encoder $V$, and coefficient vector $r$ from the validation loss $L(A, W_2'(W_1'(A), V, r), D^{(val)})$. The encoder model $V$ can be updated as:

$$V' = V - \eta_V \nabla_V L(A, W_2', D^{(val)})$$
$$= V + \eta_V \eta_{W_2} \frac{\partial}{\partial V} \Big( \big( \nabla_{W_2} \sum_{i=1}^{N^{tr}} a_i(V) \ell(A, W_2, d_i^{(tr)}) \big).$$
$$\big( \nabla_{W_2'} L(A, W_2', D^{(val)}) \big) \Big) \qquad (11)$$

Similarly, the coefficient vector $r$ can be updated as:

$$r' = r - \eta_r \nabla_r L(A, W_2', D^{(val)})$$
$$= r + \eta_r \eta_{W_2} \frac{\partial}{\partial r} \Big( \big( \nabla_{W_2} \sum_{i=1}^{N^{tr}} a_i(V) l(A, W_2, d_i^{(tr)}) \big).$$
$$\big( \nabla_{W_2'} L(A, W_2', D^{(val)}) \big) \Big) \qquad (12)$$

Next, we give the update equation for the architecture $A$.

$$A' = A - \eta_A \nabla_A L(A, W_2'(A), D^{(val)})$$
$$= A - \eta_A \Big( \frac{\partial L(A, W_2', D^{(val)})}{\partial A} +$$
$$\frac{\partial L(A, W_2', D^{(val)})}{\partial W_2'} \frac{\partial W_2'(A)}{\partial A} \Big) \qquad (13)$$

To save space, the complete derivation is not given in the paper.

$$A' = A - \eta_A \Big( \frac{\partial L(A, W_2', D^{(val)})}{\partial A} - \eta_{W_2} \Big($$
$$- \eta_{W_1} \frac{\nabla_A L(A, W_1^+, D^{(tr)}) - \nabla_A L(A, W_1^-, D^{(tr)})}{2\epsilon_1}$$
$$+ \frac{\nabla_A \sum_i a_i \big( L(A, W_2^+, d_i^{(tr)}) - L(A, W_2^-, d_i^{(tr)}) \big)}{2\epsilon_2} \Big) \Big)$$
$$\qquad (14)$$

where

$$W_1^\pm = W_1 \pm \epsilon_1 \nabla_{W_1'} \big( (\nabla_{W_2} \sum_{i=1}^{N^{(tr)}} \mathbf{a}_i(W_1') l(A, W_2, d_i^{(tr)})).$$
$$\nabla_{W_2'} L(A, W_2', D^{(val)}) \big)$$

and

$$W_2^\pm = W_2 \pm \epsilon_2 \nabla_{W_2'} L(A, W_2', D^{(val)})$$

Also, $\epsilon_1$ and $\epsilon_2$ are small scalars.

The overall algorithm of LFM when applying to other related methods can be summarised in Algorithm 1. And LFM can be applied to any differentiable NAS methods. To apply our framework to a differentiable NAS method $M$, we just need to set the architecture variable $A$ in our framework to be the search space of $M$. In other words, $M$ is a special solution in the solution space $S$ of our method.

---

**Algorithm 1: Algorithm for LFM**

---

1: **while** not converged **do**
2:     Update $W_1$
3:     Update $W_2$
4:     Update $A, V, r$ respectively
5: **end while**

---

## Experiments

### Datasets

The experiments are performed on three popular NAS datasets, namely CIFAR-10, CIFAR-100 (Krizhevsky, Hinton et al. 2009) and ImageNet (Deng et al. 2009). CIFAR-10 contains 10 classes, and CIFAR-100 contains 100 classes. Both these datasets contain 60K images each with each class having the same number of images. We split each of these datasets into a training set with 25K images, a validation set with 25K images, and a test set with 10K images. During architecture search in LFM, the training set is used as $D^{tr}$ and the validation set is used as $D^{val}$. During architecture evaluation, the learned network is trained on the combination of $D^{tr}$ and $D^{val}$. Further, ImageNet contains 1.2M training images and 50K test images with 1000 objective classes.

### Experimental Settings

Our framework is orthogonal to existing NAS approaches and can be applied to any differentiable NAS method. In our experiments, we applied LFM to DARTS (Liu, Simonyan,

| Method | Error(%) |
|---|---|
| *ResNet (He et al. 2016a) | 22.10 |
| *DenseNet (Huang et al. 2017) | 17.18 |
| *PNAS (Liu et al. 2018a) | 19.53 |
| *ENAS (Pham et al. 2018) | 19.43 |
| *AmoebaNet (Real et al. 2019b) | 18.93 |
| *GDAS (Dong and Yang 2019) | 18.38 |
| *R-DARTS (Zela et al. 2020) | 18.01±0.26 |
| *DARTS$^-$ (Chu et al. 2020) | 17.51±0.25 |
| *DARTS$^-$ (Chu et al. 2020) | 18.97±0.16 |
| *DARTS$^+$ (Liang et al. 2019b) | 17.11±0.43 |
| *DropNAS (Hong et al. 2020) | 16.39 |
| Random search | 21.92±0.34 |
| Random sampling | 21.37±0.48 |
| *DARTS-2nd (Liu, Simonyan, and Yang 2019) | 20.58±0.44 |
| LFM-DARTS-2nd-R18 (ours) | **17.65±0.45** |
| *PDARTS (Chen et al. 2021) | 17.49 |
| LFM-PDARTS-R18 (ours) | **16.44±0.11** |
| LFM-PDARTS-R34 (ours) | **15.69±0.15** |

Table 1: Test error on CIFAR-100. LFM-DARTS-2nd-R18 represents that LFM is applied onto DARTS-2nd, and ResNet-18 is used as the data encoder. Similar meanings hold for other notations in such a format. Results marked with * are obtained from Skillearn (Xie, Du, and Ban 2020). The information about parameters and search cost are shown in the supplement.

and Yang 2019) and PDARTS (Chen et al. 2021). The search spaces of these methods are composed of (dilated) separable convolutions with sizes of 3 × 3 and 5 × 5, max pooling with the size of 3 × 3, average pooling with the size of 3 × 3, identity, and zero. For the encoder $V$, Res-Nets pre-trained on Imagenet were used. Each LFM experiment was repeated three times with different random seeds. The mean and standard deviation of classification errors obtained from the three runs are reported.

**Architecture Search** During architecture search for CIFAR-10 and CIFAR-100, the architectures of $W_1$ and $W_2$ are a stack of 8 cells. Each cell consists of 7 nodes. We set the initial channel number to 16. For the architecture of the encoder model, we experimented with ResNet-18 and ResNet-34 (He et al. 2016b). The search algorithm was based on SGD, and the hyperparameters of epochs, initial learning rate, and momentum follow the original implementation of the respective DARTS (Liu, Simonyan, and Yang 2019) and PDARTS (Chen et al. 2021). We use a batch size of 64 for both DARTS and PDARTS. LFM-PDARTS uses first order approximations to be consistent with the original implementation in PDARTS (Chen et al. 2021). The LFM experiments in this paper use A100 for DARTS and A40 for PDARTS.

**Architecture Evaluation** During architecture evaluation for CIFAR-10 and CIFAR-100, a larger network of each category-specific model is formed by stacking 20 copies of the searched cell. The initial channel number is set to 36. We trained the network with a batch size of 96, an epoch num-

| Method | Error(%) |
|---|---|
| *DenseNet (Huang et al. 2017) | 3.46 |
| *HierEvol (Liu et al. 2018c) | 3.75±0.12 |
| *PNAS (Liu et al. 2018a) | 3.41±0.09 |
| *NASNet-A (Zoph et al. 2018) | 2.65 |
| *AmoebaNet-B (Real et al. 2019b) | 2.55±0.05 |
| *R-DARTS (Zela et al. 2020) | 2.95±0.21 |
| *GTN (Such et al. 2019) | 2.92±0.06 |
| *BayesNAS (Zhou et al. 2019) | 2.81±0.04 |
| *MergeNAS (Wang et al. 2020) | 2.73±0.02 |
| *NoisyDARTS (Chu, Zhang, and Li 2020) | 2.70±0.23 |
| *ASAP (Noy et al. 2020) | 2.68±0.11 |
| *SDARTS (Chen and Hsieh 2020) | 2.61±0.02 |
| *DropNAS (Hong et al. 2020) | 2.58±0.14 |
| *DrNAS (Chen et al. 2020) | 2.54±0.03 |
| Random search | 3.07±0.17 |
| Random sampling | 2.75±0.09 |
| *DARTS-2nd (Liu, Simonyan, and Yang 2019) | 2.76±0.09 |
| LFM-DARTS-2nd-R18 (ours) | **2.70±0.06** |
| *PDARTS (Chen et al. 2021) | 2.50 |
| LFM-PDARTS-R18 (ours) | **2.46±0.04** |

Table 2: Test error on CIFAR-10. Results marked with * are obtained from Skillearn (Xie, Du, and Ban 2020). The other notations are same as described in Table 1. The information about parameters, search cost and more compared methods are shown in the supplement.

ber of 600, on a single Tesla v100 GPU. On ImageNet, we evaluate the architectures searched on CIFAR-10 or CIFAR-100. In either type, 14 copies of optimally searched cells are stacked into a large network, which was trained using two Tesla A100 GPUs on the 1.2M training images, with a batch size of 1024 and an epoch number of 250. The initial channel number is set to 48.

The LFM method is used to learn the architecture $A$, while the weights $W_1$, $W_2$, $V$, and $r$ learnt during the LFM search are discarded during the architecture evaluation. All the architecture evaluations are run using the same standardized setup as described in the above paragraph. This results in a fair comparison between architectures learnt from different methods, as all the models during evaluation have same number of parameters and hyper-parameters such as epochs, learning rate, and batch size.

## Results

**Result of classification in different datasets** The results of the classification error(%) of different NAS methods on CIFAR-100 are showed in Table 1. We make the following observations from this table:

- When LFM is applied DARTS-2nd (second-order approximation) and PDARTS, the classification errors of these methods are reduced significantly. For example, when LFM is applied to DARTS-2nd, the error reduces from 20.58% to 17.70%. In PDARTS, the error reduces from 17.49% to 16.44% (when using ResNet-18 as encoder $V$). This shows the effectiveness of our method in improving the performance of architecture search. In the

| Method | Top-1 Error (%) | Top-5 Error (%) |
|---|---|---|
| *Inception-v1 (Szegedy et al. 2015) | 30.2 | 10.1 |
| *MobileNet (Howard et al. 2017) | 29.4 | 10.5 |
| *ShuffleNet $2\times$ (v1) (Zhang et al. 2018) | 26.4 | 10.2 |
| *ShuffleNet $2\times$ (v2) (Ma et al. 2018) | 25.1 | 7.6 |
| *NASNet-A (Zoph et al. 2018) | 26.0 | 8.4 |
| *PNAS (Liu et al. 2018a) | 25.8 | 8.1 |
| *MnasNet-92 (Tan et al. 2019) | 25.2 | 8.0 |
| *AmoebaNet-C (Real et al. 2019b) | 24.3 | 7.6 |
| *SNAS-CIFAR10 (Xie et al. 2019) | 27.3 | 9.2 |
| *PARSEC-CIFAR10 (Casale, Gordon, and Fusi 2019) | 26.0 | 8.4 |
| *DSNAS-ImageNet (Hu et al. 2020) | 25.7 | 8.1 |
| *SDARTS-ADV-CIFAR10 (Chen and Hsieh 2020) | 25.2 | 7.8 |
| *FairDARTS-ImageNet (Chu et al. 2019) | 24.4 | 7.4 |
| *DrNAS-ImageNet (Chen et al. 2020) | 24.2 | 7.3 |
| *ProxylessNAS-ImageNet (Cai, Zhu, and Han 2019) | 24.9 | 7.5 |
| *GDAS-CIFAR10 (Dong and Yang 2019) | 26.0 | 8.5 |
| *DARTS2nd-CIFAR10 (Liu, Simonyan, and Yang 2019) | 26.7 | 8.7 |
| LFM-DARTS-2nd-CIFAR10 (ours) | **25.12** | **7.65** |
| *PDARTS (CIFAR10) (Chen et al. 2021) | 24.4 | 7.4 |
| LFM-PDARTS-CIFAR10 (ours) | **24.14** | **6.85** |
| *PDARTS (CIFAR100) (Chen et al. 2021) | 24.7 | 7.5 |
| LFM-PDARTS-CIFAR100 (ours) | **24.11** | **6.70** |

Table 3: Top-1 and top-5 classification errors on the test set of ImageNet. Results marked with * are obtained from Skillearn (Xie, Du, and Ban 2020). From top to bottom, on the first three blocks are 1) networks manually designed by humans; 2) non-gradient based NAS methods; and 3) gradient-based NAS methods. Rest of the notations follow Tables 1, 2. The information about parameters, search cost and more compared methods are shown in the supplement.

baseline NAS approaches, all the training examples have the same weight, which implicitly implies that all examples are equally difficult to learn. The learner, in this case, can give a good performance by performing well on the majority of easy examples and ignoring the minority of difficult examples. In contrast, our method re-weights the training example at each stage based on the current capability of the learner, giving more weight to examples that are difficult to learn, which is a more realistic scenario. The learner gains the ability to deal with more challenging cases in the unseen data.

- LFM-PDARTS-R34 outperforms LFM-PDARTS-R18 by $0.75\%$, where the former uses ResNet-34 as the image encoder, while the latter uses ResNet-18. ResNet-34 is a deeper and more powerful data encoder than ResNet-18. This shows that mapping the validation examples to similar training examples is a core component contributing to the effectiveness of our proposed LFM method.

- LFM-PDARTS-R34 achieves the best performance among all methods, which demonstrates the effectiveness of applying LFM to differentiable NAS methods and improving their performance. To the best of our knowledge, LFM-PDARTS-R34 is the new SOTA on CIFAR-100.

The results of the classification error(%) of different NAS methods on CIFAR-10 are showed in Table 2. As can be seen, LFM applied to DARTS-2nd and PDARTS reduces the errors of these baselines by roughly $0.05\%$. This further demonstrates the effectiveness of our method.

The results of the classification error(%) - top-1 and top-5 of different NAS methods on ImageNet are showed in Table 3. In methods LFM-DARTS-2nd-CIFAR10 and LFM-PDARTS-CIFAR10, the architecture searched on CIFAR-10 is evaluated on ImageNet, whereas in LFM-PDARTS-CIFAR100, the architecture searched on CIFAR-100 is evaluated on Imagenet. The LFM-DARTS-2nd-CIFAR10 outperforms the baseline DARTS-2nd-CIFAR10 by $1.6\%$, while LFM-PDARTS-CIFAR100 outperforms its corresponding baseline by $0.6\%$, and LFM-PDARTS-CIFAR10 by $0.3\%$. As shown, the LFM methods outperform their corresponding baselines, and therefore, demonstrate our method's effectiveness.

### Ablation Studies

**Ablation 1** Our method introduces three important components to the re-weighting parameter $a_i$ namely: $x$, $u$, and $z$. In this study, we demonstrate the effect of ablating each component. We perform the experiment of CIFAR-100 and set the encoder to ResNet-18. The other details are are kept same as the base experiments described in earlier sections. The performances of the ablated models are shown in 2. These results demonstrate the effectiveness and necessity of measuring mistakes $u$ on validation examples, calculating visual similarity $x$ and label similarity $z$ between training
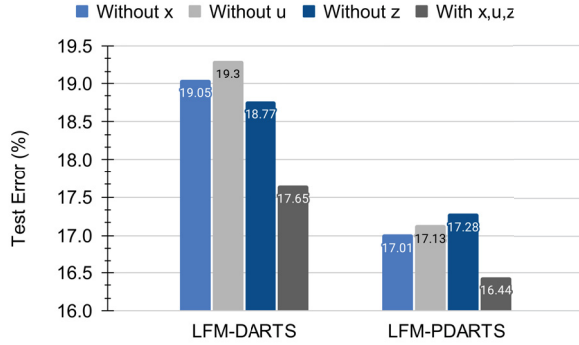
and validation examples.



Figure 2: Ablation on components of $a_i$. The left bar column shows the comparison of ablated models 1) without x, 2) without u, 3) without z , 4) and the full models.

**Ablation 2**  For visual similarity, we use dot product attention, which has shown broad effectiveness in many applications and is simple to use. To demonstrate its effectiveness in LFM, we compared with other metrics such as cosine similarity and L2 distance. The other details are are kept same as the base experiments described in earlier sections. Results are shown in Figure 3. Dot product attention used in our framework works better than the other two metrics.
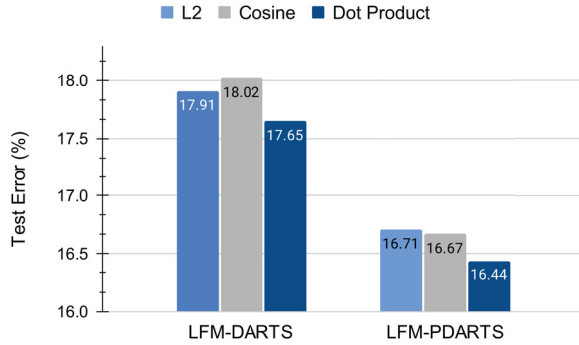


Figure 3: Comparison of different visual similarity metrics.

**Ablation 3**  Since we use the data encoder that is pretrained on ImageNet, it may provide an unfair advantage to our method in the CIFAR100 experiment since the encoder is exposed to more data. We set experiments to restrict our method to only using the CIFAR100 dataset and check its performance. We train the encoder solely on CIFAR-100, without using ImageNet pre-training. The results (on Cifar100) are given in Table 4 below. Removing ImageNet pre-training does not increase test errors significantly, showing that the improvement achieved by our method over DART-S/PDARTS is not due to ImageNet pretraining.

| Method | Test error(%) |
|---|---|
| LFM+DARTS, no INP | 17.82±0.39 |
| LFM+DARTS+INP | 17.65±0.45 |
| DARTS | 20.58±0.44 |
| LFM+PDARTS, no INP | 16.51±0.13 |
| LFM+PDARTS+INP | 16.44±0.11 |
| PDARTS | 17.49 |

Table 4: Comparisons on different pre-training datasets of the encoder. InP means ImageNet pretrain.

## Limitations and Future Work

Our method requires the use of two learners that have similar learning capabilities, so that one can learn from the mistakes of other. This increases the memory requirements and makes the learning slow compared to the vanilla approaches. As future work, we explore to reduce memory cost during architecture search by parameter-sharing between the three models $W_1$, $W_2$, and $V$. For $W_1$ and $W_2$, we let them share the same convolutional layers, but have different classification heads. For $V$, we replace ResNet-18 with $W_1$. As shown in Table 5 that via parameter sharing (PS), the memory and computation costs of our method are reduced to a level similar to vanilla DARTS and PDARTS, while our method still achieves significantly lower test errors than DARTS and PDARTS. A future work direction is to improve memory usage while keeping the full performance of the LFM method. Another direction is to extend the applicability of LFM to other meta-learning tasks such as data re-weighting, or more complex tasks like semantic segmentation. Further, LFM can be extended to language modeling tasks as well.

| Method | Test error (%) | Memory (MiB) | Cost (days) |
|---|---|---|---|
| LFM+DARTS, no PS | 17.65±0.45 | 23,702 | 5.4 |
| LFM+DARTS+PS | 18.77±0.31 | 12,138 | 1.6 |
| DARTS | 20.58±0.44 | 11,053 | 1.5 |
| LFM+PDARTS, no PS | 16.44±0.11 | 20,744 | 2.0 |
| LFM+PDARTS+PS | 16.83±0.08 | 10,582 | 0.3 |
| PDARTS | 17.49 | 9,659 | 0.3 |

Table 5: Test error(%), memory cost (MiB) and computation cost (GPU days) of different models on CIFAR-100.

## Conclusions

In this paper, we propose a novel strategy to improve the performance of differential Neural Architecture Search approaches - Learning from Mistakes (LFM) inspired by the astounding ability of humans to learn from feedback on their current performance. The learner model continuously focuses more on the mistakes and improves its learning ability on more challenging examples. We propose a multi-level optimization framework to formalize LFM and provide an efficient solution to the problem. We apply LFM to NAS on datasets of CIFAR-10, CIFAR-100, and Imagenet - pushing the frontiers on NAS research.

# References

Bachem, O.; Lucic, M.; and Krause, A. 2017. Practical coreset constructions for machine learning. *arXiv preprint arXiv:1703.06476*.

Boutsidis, C.; Mahoney, M. W.; and Drineas, P. 2009. An improved approximation algorithm for the column subset selection problem. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, 968–977. SIAM.

Cai, H.; Zhu, L.; and Han, S. 2019. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In *ICLR*.

Casale, F. P.; Gordon, J.; and Fusi, N. 2019. Probabilistic Neural Architecture Search. *CoRR*, abs/1902.05116.

Chen, X.; and Hsieh, C. 2020. Stabilizing Differentiable Architecture Search via Perturbation-based Regularization. *CoRR*, abs/2002.05283.

Chen, X.; Wang, R.; Cheng, M.; Tang, X.; and Hsieh, C. 2020. DrNAS: Dirichlet Neural Architecture Search. *CoRR*, abs/2006.10355.

Chen, X.; Xie, L.; Wu, J.; and Tian, Q. 2021. Progressive darts: Bridging the optimization gap for nas in the wild. *International Journal of Computer Vision*, 129(3): 638–655.

Chu, X.; Wang, X.; Zhang, B.; Lu, S.; Wei, X.; and Yan, J. 2020. DARTS-: Robustly Stepping out of Performance Collapse Without Indicators. *CoRR*, abs/2009.01027.

Chu, X.; Wang, X.; Zhang, B.; Lu, S.; Wei, X.; and Yan, J. 2021. {DARTS}-: Robustly Stepping out of Performance Collapse Without Indicators. In *International Conference on Learning Representations*.

Chu, X.; Zhang, B.; and Li, X. 2020. Noisy Differentiable Architecture Search. *CoRR*, abs/2005.03566.

Chu, X.; Zhou, T.; Zhang, B.; and Li, J. 2019. Fair DARTS: Eliminating Unfair Advantages in Differentiable Architecture Search. *CoRR*, abs/1911.12126.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 248–255. Ieee.

Deshpande, A.; and Rademacher, L. 2010. Efficient volume sampling for row/column subset selection. In *2010 ieee 51st annual symposium on foundations of computer science*, 329–338. IEEE.

Dong, X.; and Yang, Y. 2019. Searching for a Robust Neural Architecture in Four GPU Hours. In *CVPR*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016a. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016b. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778.

Hong, W.; Li, G.; Zhang, W.; Tang, R.; Wang, Y.; Li, Z.; and Yu, Y. 2020. DropNAS: Grouped Operation Dropout for Differentiable Architecture Search. In *IJCAI*.

Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; and Adam, H. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CoRR*, abs/1704.04861.

Hu, S.; Xie, S.; Zheng, H.; Liu, C.; Shi, J.; Liu, X.; and Lin, D. 2020. DSNAS: Direct Neural Architecture Search Without Parameter Retraining. In *CVPR*.

Huang, G.; Liu, Z.; van der Maaten, L.; and Weinberger, K. Q. 2017. Densely Connected Convolutional Networks. In *CVPR*.

Krizhevsky, A.; Hinton, G.; et al. 2009. Learning multiple layers of features from tiny images. *Citeseer*.

Liang, H.; Zhang, S.; Sun, J.; He, X.; Huang, W.; Zhuang, K.; and Li, Z. 2019a. Darts+: Improved differentiable architecture search with early stopping. *arXiv preprint arXiv:1909.06035*.

Liang, H.; Zhang, S.; Sun, J.; He, X.; Huang, W.; Zhuang, K.; and Li, Z. 2019b. DARTS+: Improved Differentiable Architecture Search with Early Stopping. *CoRR*, abs/1909.06035.

Liu, C.; Zoph, B.; Neumann, M.; Shlens, J.; Hua, W.; Li, L.; Fei-Fei, L.; Yuille, A. L.; Huang, J.; and Murphy, K. 2018a. Progressive Neural Architecture Search. In *ECCV*.

Liu, H.; Simonyan, K.; Vinyals, O.; Fernando, C.; and Kavukcuoglu, K. 2018b. Hierarchical Representations for Efficient Architecture Search. In *International Conference on Learning Representations*.

Liu, H.; Simonyan, K.; Vinyals, O.; Fernando, C.; and Kavukcuoglu, K. 2018c. Hierarchical Representations for Efficient Architecture Search. In *ICLR*.

Liu, H.; Simonyan, K.; and Yang, Y. 2019. DARTS: Differentiable Architecture Search. In *International Conference on Learning Representations*.

Luong, M.-T.; Pham, H.; and Manning, C. D. 2015. Effective Approaches to Attention-based Neural Machine Translation. In *Empirical Methods in Natural Language Processing (EMNLP)*, 1412–1421. Lisbon, Portugal: Association for Computational Linguistics.

Ma, N.; Zhang, X.; Zheng, H.; and Sun, J. 2018. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In *ECCV*.

Noy, A.; Nayman, N.; Ridnik, T.; Zamir, N.; Doveh, S.; Friedman, I.; Giryes, R.; and Zelnik, L. 2020. ASAP: Architecture Search, Anneal and Prune. In *AISTATS*.

Pham, H.; Guan, M. Y.; Zoph, B.; Le, Q. V.; and Dean, J. 2018. Efficient Neural Architecture Search via Parameter Sharing. In *ICML*.

Real, E.; Aggarwal, A.; Huang, Y.; and Le, Q. V. 2019a. Regularized Evolution for Image Classifier Architecture Search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01): 4780–4789.

Real, E.; Aggarwal, A.; Huang, Y.; and Le, Q. V. 2019b. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, 4780–4789.

Ren, M.; Zeng, W.; Yang, B.; and Urtasun, R. 2018. Learning to reweight examples for robust deep learning. *arXiv preprint arXiv:1803.09050*.

Shu, J.; Xie, Q.; Yi, L.; Zhao, Q.; Zhou, S.; Xu, Z.; and Meng, D. 2019. Meta-weight-net: Learning an explicit mapping for sample weighting. In *Advances in Neural Information Processing Systems*, 1919–1930.

Such, F. P.; Rawal, A.; Lehman, J.; Stanley, K. O.; and Clune, J. 2019. Generative Teaching Networks: Accelerating Neural Architecture Search by Learning to Generate Synthetic Training Data. *CoRR*, abs/1912.07768.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *CVPR*.

Tan, M.; Chen, B.; Pang, R.; Vasudevan, V.; Sandler, M.; Howard, A.; and Le, Q. V. 2019. MnasNet: Platform-Aware Neural Architecture Search for Mobile. In *CVPR*.

Wang, X.; Xue, C.; Yan, J.; Yang, X.; Hu, Y.; and Sun, K. 2020. MergeNAS: Merge Operations into One for Differentiable Architecture Search. In *IJCAI*.

Xie, P.; Du, X.; and Ban, H. 2020. Skillearn: Machine Learning Inspired by Humans' Learning Skills. *arXiv:2012.04863 [cs]*. ArXiv: 2012.04863.

Xie, S.; Zheng, H.; Liu, C.; and Lin, L. 2019. SNAS: stochastic neural architecture search. In *ICLR*.

Xu, Y.; Xie, L.; Dai, W.; Zhang, X.; Chen, X.; Qi, G.-J.; Xiong, H.; and Tian, Q. 2021. Partially-Connected Neural Architecture Search for Reduced Computational Redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Zela, A.; Elsken, T.; Saikia, T.; Marrakchi, Y.; Brox, T.; and Hutter, F. 2020. Understanding and Robustifying Differentiable Architecture Search. In *ICLR*.

Zhang, X.; Zhou, X.; Lin, M.; and Sun, J. 2018. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In *CVPR*.

Zhou, H.; Yang, M.; Wang, J.; and Pan, W. 2019. BayesNAS: A Bayesian Approach for Neural Architecture Search. In *ICML*.

Zoph, B.; and Le, Q. V. 2017. Neural Architecture Search with Reinforcement Learning. In *ICLR*.

Zoph, B.; Vasudevan, V.; Shlens, J.; and Le, Q. V. 2018. Learning transferable architectures for scalable image recognition. In *CVPR*.