Learning by Passing Tests, with Application to Neural Architecture Search

Xuefeng Du[†] XUEFENGDU1@GMAIL.COM
Haochen Zhang[†] ZHC12345@MAIL.USTC.EDU.CN
Pengtao Xie[*] P1XIE@ENG.UCSD.EDU
UC San Diego

Abstract

Learning through tests is a broadly used methodology in human learning and shows great effectiveness in improving learning outcome: a sequence of tests are made with increasing levels of difficulty; the learner takes these tests to identify his/her weak points in learning and continuously addresses these weak points to successfully pass these tests. We are interested in investigating whether this powerful learning technique can be borrowed from humans to improve the learning abilities of machines. We propose a novel learning approach called learning by passing tests (LPT). In our approach, a tester model creates increasingly more-difficult tests to evaluate a learner model. The learner tries to continuously improve its learning ability so that it can successfully pass however difficult tests created by the tester. We propose a multi-level optimization framework to formulate LPT, where the tester learns to create difficult and meaningful tests and the learner learns to pass these tests. We develop an efficient algorithm to solve the LPT problem. Our method is applied for neural architecture search and achieves significant improvement over state-of-the-art baselines on CIFAR-100, CIFAR-10, and ImageNet.

1. Introduction

In human learning, an effective and widely used methodology for improving learning outcome is to let the learner take increasingly more-difficult tests. To successfully pass a more challenging test, the learner needs to gain better learning ability. By progressively passing tests that have increasing levels of difficulty, the learner strengthens his/her learning capability gradually.

Inspired by this test-driven learning technique of humans, we are interested in investigating whether this methodology is helpful for improving machine learning as well. We propose a novel machine learning framework called learning by passing tests (LPT). In this framework, there is a "learner" model and a "tester" model. The tester creates a sequence of "tests" with growing levels of difficulty. The learner tries to learn better so that it can pass these increasingly more-challenging tests. Given a large collection of data examples called "test bank", the tester creates a test T by selecting a subset of examples from the test bank. The learner applies its intermediately-trained model M to make predictions on

. [†]Equal contribution.
. [*]Corresponding author.

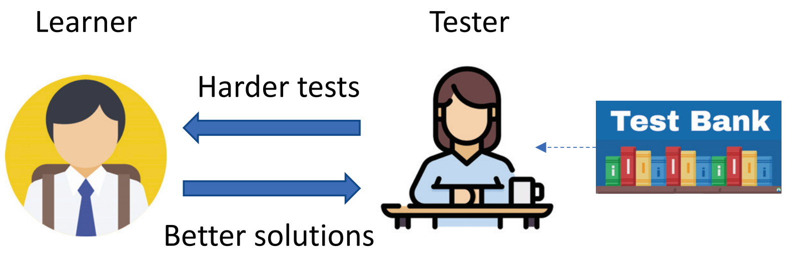

Figure 1: Learning by passing tests. A tester model creates tests with increasing levels of difficulty from a test bank to evaluate a learner model. The learner continuously improves its learning ability to deliver better solutions for passing those difficult tests.

the examples in T. The prediction error rate R reflects how difficult this test is. If the learner can make correct predictions on T, it means that T is not difficult enough. In this case, the tester will create a more challenging test T' by selecting a new set of examples from the test bank such that the new error rate R' achieved by M on T' is larger than R achieved on T. Given this more demanding test T', the learner re-learns its model to pass T', in a way that the newly-learned model M' achieves a new error rate R'' on T' where R'' is smaller than R'. This process (as illustrated in Figure 1) iterates until convergence.

In our framework, both the learner and tester perform learning. The learner learns how to best conduct a target task J_1 and the tester learns how to create difficult and meaningful tests. To encourage a created test T to be meaningful, the tester trains a model using T to perform a target task J_2. If the model performs well on J_2, it indicates that T is meaningful. The learner has two sets of learnable parameters: neural architecture and network weights. The tester has three learnable modules: data encoder, test creator, and target-task executor. Learning is organized into three stages. In the first stage, the learner trains its network weights on the training set of task J_1 with the architecture fixed. In the second stage, the tester trains its data encoder and target-task executor on a created test to perform the target task J_2, with the test creator fixed. In the third stage, the learner updates its model architecture by minimizing the predictive loss L on the test created by the tester; the tester updates its test creator by maximizing L and minimizing the loss on the validation set of J_2. The three stages are performed jointly end-to-end in a multi-level optimization framework, where different stages influence each other. We apply our method for neural architecture search (Zoph and Le, 2017; Liu et al., 2019; Real et al., 2019) in image classification tasks on CIFAR-100, CIFAR-10, and ImageNet (Deng et al., 2009). Our method achieves significant improvement over state-of-the-art baselines.

The major contributions of this paper are as follows:

- Inspired by the test-driven learning technique of humans, we propose a novel ML approach called learning by passing tests (LPT). In our approach, a tester model creates increasingly more-difficult tests to evaluate a learner model. The learner tries to continuously improve its learning ability so that it can successfully pass however difficult tests created by the tester.

- We propose a multi-level optimization framework to formulate LPT where a learner learns to pass tests and a tester learns to create difficult and meaningful tests.
- We develop an efficient algorithm to solve LPT.
- We apply our approach to neural architecture search and achieve significant improvement on CIFAR-100, CIFAR-10, and ImageNet.

2. Related Works

Neural Architecture Search (NAS). NAS has achieved remarkable progress recently, which aims at searching for optimal architectures of neural networks to achieve the best predictive performance. In general, there are three paradigms of methods in NAS: reinforcement learning based approaches (Zoph and Le, 2017; Pham et al., 2018; Zoph et al., 2018), evolutionary algorithm based approaches (Liu et al., 2018b; Real et al., 2019), and differentiable approaches (Liu et al., 2019; Cai et al., 2019; Xie et al., 2019). In RL-based approaches, a policy is learned to iteratively generate new architectures by maximizing a reward which is the accuracy on the validation set. Evolutionary learning approaches represent the architectures as individuals in a population. Individuals with high fitness scores (validation accuracy) have the privilege to generate offspring, which replaces individuals with low fitness scores. Differentiable approaches adopt a network pruning strategy. On top of an over-parameterized network, the weights of connections between nodes are learned using gradient descent. Then weights close to zero are pruned later on. There have been many efforts devoted to improving differentiable NAS methods. In P-DARTS (Chen et al., 2019), the depth of searched architectures is allowed to grow progressively during the training process. Search space approximation and regularization approaches are developed to reduce computational overheads and improve search stability. PC-DARTS (Xu et al., 2020) reduces the redundancy in exploring the search space by sampling a small portion of a super network. Operation search is performed in a subset of channels with the held-out part bypassed in a shortcut. Our proposed LPT framework is orthogonal to existing NAS approaches and can be applied to any differentiable NAS methods.

Adversarial Learning. Our formulation involves a min-max optimization problem, which is analogous to that in adversarial learning (Goodfellow et al., 2014a) for data generation (Goodfellow et al., 2014a; Yu et al., 2017), domain adaptation (Ganin and Lempitsky, 2015), adversarial attack and defence (Goodfellow et al., 2014b), etc. Adversarial learning (Goodfellow et al., 2014a) has been widely applied to 1) data generation (Goodfellow et al., 2014a; Yu et al., 2017) where a discriminator tries to distinguish between generated images and real images and a generator is trained to generate realistic data by making such a discrimination difficult to achieve; 2) domain adaptation (Ganin and Lempitsky, 2015) where a discriminator tries to differentiate between source images and target images while the feature learner learns representations which make such a discrimination unachievable; 3) adversarial attack and defence (Goodfellow et al., 2014b) where an attacker adds small perturbations to the input data to alter the prediction outcome and the defender trains the model in a way that the prediction outcome remains the same given perturbed inputs. Different from these existing works, in our work, a tester aims to create harder tests to "fail" the learner while the learner learns to "pass" however hard tests created by the tester. Shu

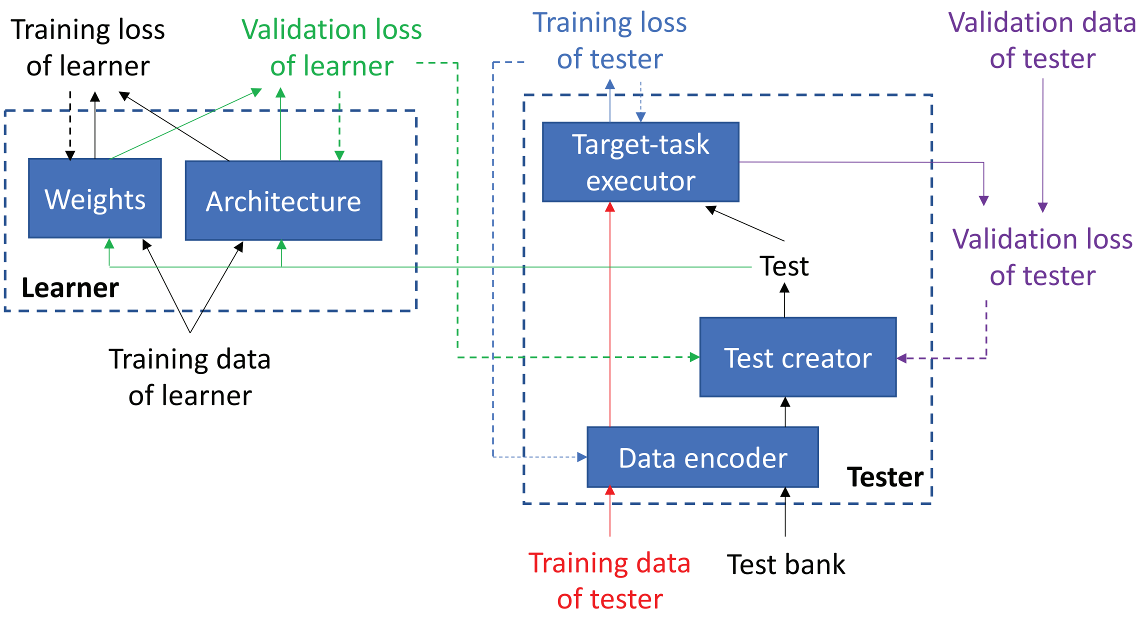

Figure 2: Learning by passing tests. The solid arrows denote the process of making predictions and calculating losses. The dotted arrows denote the process of updating learnable parameters by minimizing corresponding losses.

et al. (2020) proposed to use an adversarial examiner to identify the weakness of a trained model. Our work differs from this work in that we progressively re-train a learner model based on how it performs on the tests that are created dynamically by a tester model while the learner model in (Shu et al., 2020) is fixed and not affected by the examination results. Such et al. (2019) proposed to learn a generative adversarial network (Goodfellow et al., 2014a) to create synthetic examples which are used to train an NAS model. Our work differs from this work in that we use selected validation examples to validate the model while Such et al. (2019) use synthesized example to train the model.

Curriculum Learning. Our work is also related to curriculum learning (CL) (Bengio et al., 2009; Kumar et al., 2010; Jiang et al., 2014; Matiisen et al., 2019). In CL, a sequence of training datasets with increasing levels of difficulty is used for model training, from easy to difficult. Our work differs from these previous works in that: our work dynamically selects more-difficult data examples for model evaluation while previous works select data examples for model training.

3. Methods

In this section, we propose a framework to perform learning by passing tests (LPT) (as shown in Figure 2) and develop an optimization algorithm for solving the LPT problem.

Table 1: Notations in Learning by Passing Tests

Notation	Meaning
A	Architecture of the learner
W	Network weights of the learner
E	Data encoder of the tester
C	Test creator of the tester
X	Target-task executor of the tester
$D_{ln}^{(\mathrm{tr})}$	Training data of the learner
$D_{tt}^{(\mathrm{tr})}$	Training data of the tester
$D_{tt}^{(\mathrm{val})}$	Validation data of the tester
D_b	Test bank

3.1. Learning by Passing Tests

In our framework, there is a learner model and a tester model, where the learner studies how to perform a target task J_1 such as classification, regression, etc. The eventual goal is to make the learner achieve a better learning outcome with help from the tester. There is a collection of data examples called "test bank". The tester creates a test by selecting a subset of examples from the test bank. Given a test T, the learner applies its intermediately-trained model M to make predictions on T and measures the prediction error rate R. From the perspective of the tester, R indicates how difficult the test T is. If R is small, it means that the learner can easily pass this test. Under such circumstances, the tester will create a more difficult test T' which renders the new error rate R' achieved by M on T' is larger than R. From the learner's perspective, R' indicates how well the learner performs on the test. Given this more difficult test T', the learner refines its model to pass this new test. It aims to learn a new model M' such that the newer error rate R'' achieved by M' on T' is smaller than R'. This process iterates until an equilibrium is reached. In addition to being difficult, the created test should be meaningful as well. It is possible that the test bank contains poor-quality examples where the class labels may be incorrect or the input data instances are outliers. Using an unmeaningful test containing poor-quality examples to guide the learning of the learner may render the learner to overfit these bad-quality examples and generalize poorly on unseen data. To address this problem, we encourage the tester to generate meaningful tests by leveraging the generated tests to perform a target task J_2. Specifically, the tester uses examples in the test to train a model for performing J_2. If the performance (e.g., accuracy) P achieved by this model in conducting J_2 is high, the test is considered to be meaningful. The tester aims to create a test that can yield a high P.

In our framework, both the learner and the tester performs learning. The learner studies how to best fulfill the target task J_1. The tester studies how to create tests that are difficult and meaningful. In the learner' model, there are two sets of learnable parameters: model architecture and network weights. The architecture and weights are both used to make predictions in J_1. The tester's model performs two tasks simultaneously: creating tests and performing another target-task J_2. The model has three learnable modules: data encoder, test creator, and target-task executor, where the test creator performs the task of generating tests and the target-task executor conducts J_2. The test creator and target-task executor share the same data encoder. The data encoder takes a data example d as input and generates a latent representation for this example. Then the representation is fed into the test creator which determines whether d should be selected into the test. The representation is also fed into the target-task executor which performs prediction on d during performing the target task J_2.

In our framework, the learning of the learner and the tester is organized into three stages. In the first stage, the learner learns its network weights W by minimizing the training loss $L(A, W, D_{ln}^{(\mathrm{tr})})$ defined on the training data $D_{ln}^{(\mathrm{tr})}$ in the task J_1. The architecture A is used to define the training loss, but it is not learned at this stage. If A is learned by minimizing this training loss, a trivial solution will be yielded where A is very large and complex that it can perfectly overfit the training data but will generalize poorly on unseen data. Let $W^*(A)$ denotes the optimally learned W at this stage. Note that W^* is a function of A because W^* is a function of the training loss and the training loss is a function of A. In the second stage, the tester learns its data encoder E and target-task executor X by minimizing the training loss $L(E, X, D_{tt}^{(\mathrm{tr})}) + \gamma L(E, X, \sigma(C, E, D_b))$ in the task J_2. The training loss consists of two parts. The first part $L(E, X, D_{tt}^{(\mathrm{tr})})$ is defined on the training dataset $D_{tt}^{(\mathrm{tr})}$ in J_2. The second part $L(E, X, \sigma(C, E, D_b))$ is defined on the test $\sigma(C, E, D_b)$ created by the test creator. To create a test, for each example d in the test bank D_b, it is first fed into the encoder E, then into the test creator C, which outputs a binary value indicating whether d should be selected into the test. $\sigma(C, E, D_b)$ is the collection of examples whose binary value is equal to 1. γ is a tradeoff parameter between these two parts of losses. The creator C is used to define the second-part loss, but it is not learned at this stage. Otherwise, a trivial solution will be yielded where C always sets the binary value to 0 for each test-bank example so that the second-part loss becomes 0. Let $E^*(C)$ and $X^*(C)$ denote the optimally trained E and X at this stage. Note that they are both functions of C since they are functions of the training loss and the training loss is a function of C. In the third stage, the learner learns its architecture by trying to pass the test $\sigma(C, E^*(C), D_b)$ created by the tester. Specifically, the learner aims to minimize its predictive loss on the test:

$$L(A, W^*(A), \sigma(C, E^*(C), D_b)) = \sum_{d \in \sigma(C, E^*(C), D_b)} \ell(A, W^*(A), d), \tag{1}$$

where d is an example in the test and $\ell(A, W^*(A), d)$ is the loss defined in this example. A smaller $L(A, W^*(A), \sigma(C, E^*(C), D_b))$ indicates that the learner performs well on this test.

Meanwhile, the tester learns its test creator C in a way that C can create a test with more difficulty and meaningfulness. Difficulty is measured by the learner's predictive loss $L(A, W^*(A), \sigma(C, E^*(C), D_b))$ on the test. Given a model $(A, W^*(A))$ of the learner and two tests of the same size (same number of examples): $\sigma(C_1, E^*(C_1), D_b)$ created by C_1 and $\sigma(C_2, E^*(C_2), D_b)$ created by C_2, if $L(A, W^*(A), \sigma(C_1, E^*(C_1), D_b)) > L(A, W^*(A), \sigma(C_2, E^*(C_2), D_b))$, it means that $\sigma(C_1, E^*(C_1), D_b)$ is more challenging to pass than $\sigma(C_2, E^*(C_2), D_b)$. Therefore, the tester can learn to create a more challenging test by maximizing $L(A, W^*(A), \sigma(C, E^*(C), D_b))$. A trivial solution of increasing $L(A, W^*(A), \sigma(C, E^*(C), D_b))$ is to enlarge the size of the test. But a larger size does not imply more difficulty. To discourage this degenerated solution from happening, we normalize the loss using the size of the test:

$$\frac{1}{\left|\sigma\left(C, E^*(C), D_b\right)\right|} L\left(A, W^*\left(A\right), \sigma\left(C, E^*(C), D_b\right)\right), \tag{2}$$

where $|\sigma(C, E^*(C), D_b)|$ is the cardinality of the set $\sigma(C, E^*(C), D_b)$. To measure the meaningfulness of a test, we check how well the optimally-trained task executor $X^*(C)$ and data encoder $E^*(C)$ of the tester perform on the validation data $D_{tt}^{(\text{val})}$ of the target task J_2, and the performance is measured by the validation loss: $L(E^*(C), X^*(C), D_{tt}^{(\text{val})})$. $E^*(C)$ and $X^*(C)$ are trained using the test generated by C in the second stage. If the validation loss is small, it means that the created test is helpful in training the task executor and therefore is considered as being meaningful. To create a meaningful test, the tester learns C by minimizing $L(E^*(C), X^*(C), D_{tt}^{(\text{val})})$. In sum, C is learned by maximizing $L(A, W^*(A), \sigma(C, E^*(C), D_b))/|\sigma(C, E^*(C), D_b)| - \lambda L(E^*(C), X^*(C), D_{tt}^{(\text{val})})$, where λ is a tradeoff parameter between these two objectives.

The three stages are mutually dependent: $W^*(A)$ learned in the first stage and $E^*(C)$ and $X^*(C)$ learned in the second stage are used to define the objective function in the third stage; the updated C and A in the third stage in turn change the objective functions in the first and second stage, which subsequently render $W^*(A)$, $E^*(C)$, and $X^*(C)$ to be changed. Putting these pieces together, we formulate LPT as the following multi-level optimization problem.

$$\begin{array}{l}
\max_C \min_A \quad \frac{1}{|\sigma(C,E^*(C),D_b)|} L\left(A, W^*\left(A\right), \sigma\left(C, E^*(C), D_b\right)\right) \\
\qquad\qquad -\lambda L\left(E^*(C), X^*(C), D_{tt}^{(\text{val})}\right) \text{ (Stage III)} \\
s.t. \;\; E^*(C), X^*(C) = \min_{E,X} \quad L\left(E, X, D_{tt}^{(\text{tr})}\right) + \gamma L\left(E, X, \sigma\left(C, E, D_b\right)\right) \text{ (II)} \\
\qquad W^*\left(A\right) = \min_W \quad L\left(A, W, D_{ln}^{(\text{tr})}\right) \text{ (Stage I)}
\end{array} \tag{3}$$

This formulation nests three optimization problems. On the constraints of the outer optimization problem are two inner optimization problems corresponding to the first and second learning stage. The objective function of the outer optimization problem corresponds to the third learning stage.

As of now, the test $\sigma(C, E, D_b)$ is represented as a subset, which is highly discrete and therefore difficult for optimization. To address this problem, we perform a continuous relaxation of $\sigma(C, E, D_b)$:

$$\sigma(C, E, D_b) = \{(d, f(d, C, E)) | d \in D_b\}, \tag{4}$$

where for each example d in the test bank, the original binary value indicating whether d should be selected is now relaxed to a continuous probability $f(d, C, E)$ representing how likely d should be selected. Under this relaxation, $L(E, X, \sigma(C, E, D_b))$ can be computed as follows:

$$L(E, X, \sigma(C, E, D_b)) = \sum_{d \in D_b} f(d, C, E) \ell(E, X, d), \tag{5}$$

where we calculate the loss $\ell(E, X, d)$ on each test-bank example and weigh this loss using $f(d, C, E)$. If $f(d, C, E)$ is small, it means that d is less likely to be selected into the test and its corresponding loss should be down-weighted. Similarly, $L(A, W^*(A), \sigma(C, E^*(C), D_b))$ is calculated as $\sum_{d\in D_b} f(d, C, E^*(C))\ell(A, W^*(A), d)$. And $|\sigma(C, E^*(C), D_b)|$ can be calculated as

$$|\sigma(C, E^*(C), D_b)| = \sum_{d\in D_b} f(d, C, E^*(C)). \tag{6}$$

Similar to (Liu et al., 2019), we represent the architecture A of the learner in a differentiable way. The search space of A is composed of a large number of building blocks. The output of each block is associated with a variable a indicating how important this block is. After learning, blocks whose a is among the largest are retained to form the final architecture. In this end, architecture search amounts to optimizing the set of architecture variables $A = \{a\}$.

3.2. Optimization Algorithm

In this section, we derive an optimization algorithm to solve the LPT problem. Inspired by (Liu et al., 2019), we approximate $E^*(C)$ and $X^*(C)$ using one-step gradient descent update of E and X with respect to $L(E, X, D_{tt}^{(\mathrm{tr})}) + \gamma L(E, X, \sigma(C, E, D_b))$ and approximate $W^*(A)$ using one-step gradient descent update of W with respect to $L(A, W, D_{ln}^{(\mathrm{tr})})$. Then we plug these approximations into

$$L(A, W^*(A), \sigma(C, E^*(C), D_b))/|\sigma(C, E^*(C), D_b)| - \lambda L(E^*(C), X^*(C), D_{tt}^{(\mathrm{val})}), \tag{7}$$

and perform gradient-descent update of C and A with respect to this approximated objective. In the sequel, we use $\nabla^2_{Y,X} f(X, Y)$ to denote $\frac{\partial f(X,Y)}{\partial X \partial Y}$.

Approximating $W^*(A)$ using $W' = W - \xi_{ln}\nabla_W L(A, W, D_{ln}^{(\mathrm{tr})})$ where ξ_{ln} is a learning rate and simplifying the notation of $\sigma(C, E^*(C), D_b)$ as σ, we can calculate the approximated gradient of $L(A, W^*(A), \sigma)$ w.r.t A as:

$$\begin{aligned}
&\nabla_A L(A, W^*(A), \sigma) \approx \\
&\nabla_A L\left(A, W - \xi_{ln}\nabla_W L\left(A, W, D_{ln}^{(\mathrm{tr})}\right), \sigma\right) = \\
&\nabla_A L(A, W', \sigma) - \xi_{ln}\nabla^2_{A,W} L\left(A, W, D_{ln}^{(\mathrm{tr})}\right)\nabla_{W'} L(A, W', \sigma).
\end{aligned} \tag{8}$$

The second term in the third line involves expensive matrix-vector product, whose computational complexity can be reduced by a finite difference approximation:

$$\nabla^2_{A,W} L\left(A, W, D_{ln}^{(\mathrm{tr})}\right)\nabla_{W'} L(A, W', \sigma) \approx \frac{1}{2\alpha_{ln}}\left(\nabla_A L\left(A, W^+, D_{ln}^{(\mathrm{tr})}\right) - \nabla_A L\left(A, W^-, D_{ln}^{(\mathrm{tr})}\right)\right), \tag{9}$$

where $W^\pm = W \pm \alpha_{ln}\nabla_{W'} L(A, W', \sigma)$ and α_{ln} is a small scalar that equals $0.01/\left\|\nabla_{W'} L(A, W', \sigma)\right\|_2$. We approximate $E^*(C)$ and $X^*(C)$ using the following one-step gradient descent update of E and C respectively:

$$\begin{aligned}
E' &= E - \xi_E \nabla_E[L(E, X, D_{tt}^{(\mathrm{tr})}) + \gamma L(E, X, \sigma(C, E, D_b))] \\
X' &= X - \xi_X \nabla_X[L(E, X, D_{tt}^{(\mathrm{tr})}) + \gamma L(E, X, \sigma(C, E, D_b))]
\end{aligned} \tag{10}$$

where ξ_E and ξ_X are learning rates. Plugging these approximations into the objective function in Eq.(7), we can learn C by maximizing the following objective using gradient methods:

$$L(A, W', \sigma(C, E', D_b))/|\sigma(C, E', D_b)| - \lambda L(E', X', D_{tt}^{(\text{val})}) \tag{11}$$

The derivative of the second term in this objective with respect to C can be calculated as:

$$\nabla_C L(E', X', D_{tt}^{(\text{val})}) = \frac{\partial E'}{\partial C} \nabla_{E'} L(E', X', D_{tt}^{(\text{val})}) + \frac{\partial X'}{\partial C} \nabla_{X'} L(E', X', D_{tt}^{(\text{val})}) \tag{12}$$

where

$$\begin{aligned} \frac{\partial E'}{\partial C} &= -\xi_E \gamma \nabla^2_{C,E} L(E, X, \sigma(C, E, D_b)) \\ \frac{\partial X'}{\partial C} &= -\xi_X \gamma \nabla^2_{C,X} L(E, X, \sigma(C, E, D_b)) \end{aligned} \tag{13}$$

Similar to Eq.(9), using finite difference approximation to calculate $\nabla^2_{C,E} L(E, X, \sigma(C, E, D_b)) \nabla_{E'} L(E', X', D_{tt}^{(\text{val})})$ and $\nabla^2_{C,X} L(E, X, \sigma(C, E, D_b)) \nabla_{X'} L(E', X', D_{tt}^{(\text{val})})$, we have:

$$\begin{aligned} &\nabla_C L(E', X', D_{tt}^{(\text{val})}) = \\ &-\gamma \xi_E \frac{\nabla_C L(E^+, X, \sigma(C, E^+, D_b)) - \nabla_C L(E^-, X, \sigma(C, E^-, D_b))}{2\alpha_E} - \gamma \xi_X \frac{\nabla_C L(E, X^+, \sigma(C, E, D_b)) - \nabla_C L(E, X^-, \sigma(C, E, D_b))}{2\alpha_X} \end{aligned} \tag{14}$$

where $E^{\pm} = E \pm \alpha_E \nabla_{E'} L(E', X', D_{tt}^{(\text{val})})$ and $X^{\pm} = X \pm \alpha_X \nabla_{X'} L(E', X', D_{tt}^{(\text{val})})$. For the first term $L(A, W', \sigma(C, E', D_b))/|\sigma(C, E', D_b)|$ in the objective, we can use chain rule to calculate its derivative w.r.t C, which involves calculating the derivative of $L(A, W', \sigma(C, E', D_b))$ and $|\sigma(C, E', D_b)|$ w.r.t to C. The derivative of $L(A, W', \sigma(C, E', D_b))$ w.r.t C can be calculated as:

$$\nabla_C L(A, W', \sigma(C, E', D_b)) = \frac{\partial E'}{\partial C} \nabla_{E'} L(A, W', \sigma(C, E', D_b)), \tag{15}$$

where $\frac{\partial E'}{\partial C}$ is given in Eq.(13) and $\nabla^2_{C,E} L(E, X, \sigma(C, E, D_b)) \times \nabla_{E'} L(A, W', \sigma(C, E', D_b))$ can be approximated with $\frac{1}{2\alpha_E}(\nabla_C L(E^+, X, \sigma(C, E^+, D_b)) - \nabla_C L(E^-, X, \sigma(C, E^-, D_b)))$, where $E^{\pm}$ is $E \pm \alpha_E \nabla_{E'} L(A, W', \sigma(C, E', D_b))$. The derivative of $|\sigma(C, E', D_b)| = \sum_{d \in D_b} f(d, C, E')$ w.r.t C can be calculated as

$$\sum_{d \in D_b} \nabla_C f(d, C, E') + \frac{\partial E'}{\partial C} \nabla_{E'} f(d, C, E') \tag{16}$$

where $\frac{\partial E'}{\partial C}$ is given in Eq.(13). The algorithm for solving LPT is summarized in Algorithm 1.

Algorithm 1 Optimization algorithm for learning by passing tests

```
while not converged do
    1. Update the architecture of the learner by descending the gradient calculated in Eq.(8)
    2. Update the test creator of the tester by ascending the gradient calculated in Eq.(12-16)
    3. Update the data encoder and target-task executor of the tester using Eq.(10)
    4. Update the network weights of the learner by descending $\nabla_W L(A, W, D_{ln}^{(\text{tr})})$
end
```

4. Experiments

We apply LPT for neural architecture search in image classification. Following (Liu et al., 2019), we first perform architecture search which finds an optimal cell, then perform architecture evaluation which composes multiple copies of the searched cell into a large network, trains it from scratch, and evaluates the trained model on a test set. We let the target tasks of the learner and that of the tester be the same. Please refer to the supplements for more hyperparameter settings, additional results, and significance tests of results.

4.1. Datasets

We used three datasets in the experiments: CIFAR-10, CIFAR-100, and ImageNet (Deng et al., 2009). The CIFAR-10 dataset contains 50K training images and 10K testing images, from 10 classes (the number of images in each class is equal). We split the original 50K training set into a 25K training set and a 25K validation set. In the sequel, when we mention "training set", it always refers to the new 25K training set. During architecture search, the training set is used as $D_{ln}^{(\mathrm{tr})}$ of the learner and $D_{tt}^{(\mathrm{tr})}$ of the tester. The validation set is used as the test bank D_b and the validation data $D_{tt}^{(\mathrm{val})}$ of the tester. Under such a setting, the data encoder and target-task executor of the tester are trained on a subset (which is a test) of $D_{tt}^{(\mathrm{val})}$ and validated on the entire set of $D_{tt}^{(\mathrm{val})}$. The interpretation of doing this is: we select a subset of examples from $D_{tt}^{(\mathrm{val})}$ to train a model so that it performs the best on the entire $D_{tt}^{(\mathrm{val})}$. During architecture evaluation, the combination of the training data and validation data is used to train a large network stacking multiple copies of the searched cell. The CIFAR-100 dataset contains 50K training images and 10K testing images, from 100 classes (the number of images in each class is equal). Similar to CIFAR-10, the 50K training images are split into a 25K training set and a 25K validation set. The usage of these subsets is the same as that for CIFAR-10. The ImageNet dataset contains a training set of 1.3M images and a validation set of 50K images, from 1000 object classes. The validation set is used as a test set for architecture evaluation. During architecture search, following (Xu et al., 2020), 10% of the 1.3M training images are randomly sampled to form a new training set and another 2.5% of the 1.3M training images are randomly sampled to form a new architecture validation set. The usage of the new training set and architecture validation set is the same as that in CIFAR-10. During architecture evaluation, all of the 1.3M training images are used for model training. In addition to searching architectures directly on ImageNet data, following (Liu et al., 2019), we also evaluate the architectures searched using CIFAR-10 and CIFAR-100 on ImageNet: given a cell searched using CIFAR-10 and CIFAR-100, multiple copies of it compose a large network, which is then trained on the 1.3M training data of ImageNet and evaluated on the 50K test data.

4.2. Experimental Settings

Our framework is a general one that can be used together with any differentiable search method. Specifically, we apply our framework to the following NAS methods: 1) DARTS (Liu et al., 2019), 2) P-DARTS (Chen et al., 2019), 3) DARTS^{+} (Liang et al., 2019b), 4) DARTS^{-} (Chu et al., 2020a), 5) PC-DARTS (Xu et al., 2020). The search space in these methods are similar. The candidate operations include: 3×3 and 5×5 separable convolutions, 3×3 and 5×5 dilated separable convolutions, 3×3 max pooling, 3×3 average pooling, identity, and zero. In LPT, the network of the learner is a stack of multiple cells, each consisting of 7 nodes. For the data encoder of the tester, we tried ResNet-18 and ResNet-50 (He et al., 2016b). For the test creator and target-task executor, they are set to one feed-forward layer. λ and γ are tuned using a 5k held-out dataset in $\{0.1, 0.5, 1, 2, 3\}$. In most experiments, λ and γ are set to 1 except for P-DARTS and PC-DARTS. For P-DARTS, λ, γ are set to $0.5, 1$ for CIFAR-10 and $1, 0.5$ for CIFAR-100. For PC-DARTS, we use $\lambda = 3, \gamma = 1$ and $\lambda = 0.1, \gamma = 1$ for CIFAR-10 and CIFAR-100, respectively.

For CIFAR-10 and CIFAR-100, during architecture search, the learner's network is a stack of 8 cells, with the initial channel number set to 16. The search is performed for 50 epochs, with a batch size of 64. The hyperparameters for the learner's architecture and weights are set in the same way as DARTS, P-DARTS, DARTS$^+$, and DARTS$^-$. The data encoder and target-task executor of the tester are optimized using SGD with a momentum of 0.9 and a weight decay of 3e-4. The initial learning rate is set to 0.025 with a cosine decay scheduler. The test creator is optimized with the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 3e-4 and a weight decay of 1e-3. During architecture evaluation, 20 copies of the searched cell are stacked to form the learner's network, with the initial channel number set to 36. The network is trained for 600 epochs with a batch size of 96 (for both CIFAR-10 and CIFAR-100). The experiments are performed on a single Tesla v100. For ImageNet, following (Liu et al., 2019), we take the architecture searched on CIFAR-10 and evaluate it on ImageNet. We stack 14 cells (searched on CIFAR-10) to form a large network and set the initial channel number as 48. The network is trained for 250 epochs with a batch size of 1024 on 8 Tesla v100s. Each experiment on LPT is repeated for ten times with the random seed to be from 1 to 10. We report the mean and standard deviation of results obtained from the 10 runs.

4.3. Results

Table 2 shows the classification error (%), number of weight parameters (millions), and search cost (GPU days) of different NAS methods on CIFAR-100. From this table, we make the following observations. **First**, when our method LPT is applied to different NAS baselines including DARTS-1st (first order approximation), DARTS-2nd (second order approximation), DARTS$^-$ (our run), DARTS$^+$, PC-DARTS, and P-DARTS, the classification errors of these baselines can be significantly reduced. For example, applying our method to P-DARTS, the error reduces from 17.49% to 16.28%. Applying our method to DARTS-2nd, the error reduces from 20.58% to 18.40%. This demonstrates the effectiveness of our method in searching for a better architecture. In our method, the learner continuously improves its architecture by passing the tests created by the tester with increasing levels of difficulty. These tests can help the learner to identify the weakness of its architecture and provide guidance on how to improve it. Our method creates a new test on the fly based on how the learner performs in the previous round. From the test bank, the tester selects a subset of difficult examples to evaluate the learner. This new test poses a greater challenge to the learner and encourages the learner to improve its architecture so that it can overcome the new challenge. In contrast, in baseline NAS approaches, a single fixed

Table 2: Results on CIFAR-100, including classification error (%) on the test set, number of parameters (millions) in the searched architecture, and search cost (GPU days). LPT-R18-DARTS-1st denotes that our method LPT is applied to the search space of DARTS. Similar meanings hold for other notations in such a format. R18 and R50 denote that the data encoder of the tester in LPT is set to ResNet-18 and ResNet-50 respectively. DARTS-1st and DARTS-2nd denotes that first order and second order approximation is used in DARTS. * means the results are taken from DARTS$^-$ (Chu et al., 2020a). † means we re-ran this method for 10 times. Δ means the algorithm ran for 600 epochs instead of 2000 epochs in the architecture evaluation stage, to ensure a fair comparison with other methods (where the epoch number is 600). The search cost is measured by GPU days on a Tesla v100.

Method	Error(%)	Param(M)	Cost
*ResNet (He et al., 2016a)	22.10	1.7	-
*DenseNet (Huang et al., 2017)	17.18	25.6	-
*PNAS (Liu et al., 2018a)	19.53	3.2	150
*ENAS (Pham et al., 2018)	19.43	4.6	0.5
*AmoebaNet (Real et al., 2019)	18.93	3.1	3150
*GDAS (Dong and Yang, 2019)	18.38	3.4	0.2
*R-DARTS (Zela et al., 2020)	18.01±0.26	-	1.6
*DropNAS (Hong et al., 2020)	16.39	4.4	0.7
†DARTS-1st (Liu et al., 2019)	20.52±0.31	1.8	0.4
LPT-R18-DARTS-1st (ours)	**19.11**±0.11	2.1	0.6
*DARTS-2nd (Liu et al., 2019)	20.58±0.44	1.8	1.5
LPT-R18-DARTS-2nd (ours)	19.47±0.20	2.1	1.8
LPT-R50-DARTS-2nd (ours)	**18.40**±0.16	2.5	2.0
*DARTS$^-$ (Chu et al., 2020a)	17.51±0.25	3.3	0.4
†DARTS$^-$ (Chu et al., 2020a)	18.97±0.16	3.1	0.4
LPT-R18-DARTS$^-$ (ours)	18.28±0.14	3.4	0.6
ΔDARTS$^+$ (Liang et al., 2019a)	17.11±0.43	3.8	0.2
LPT-R18-DARTS$^+$ (ours)	**16.58**±0.19	3.7	0.3
†PC-DARTS (Xu et al., 2020)	17.96±0.15	3.9	0.1
LPT-R18-PC-DARTS (ours)	17.04±0.05	3.6	0.1
LPT-R50-PC-DARTS (ours)	**16.97**±0.21	4.0	0.1
*P-DARTS (Chen et al., 2019)	17.49	3.6	0.3
LPT-R18-P-DARTS (ours)	**16.28±0.10**	3.8	0.5
LPT-R50-P-DARTS (ours)	16.38±0.07	3.6	0.5

validation set is used to evaluate the learner. The learner can achieve a good performance via "cheating": focusing on performing well on the majority of easy examples and ignoring the minority of difficult examples. As a result, the learner's architecture does not have the ability to deal with challenging cases in the unseen data. **Second**, LPT-R50-DARTS-2nd

Table 3: Results on CIFAR-10. * means the results are taken from DARTS$^-$ (Chu et al., 2020a), NoisyDARTS (Chu et al., 2020b), and DrNAS (Chen et al., 2020). The rest notations are the same as those in Table 2.

Method	Error(%)	Param(M)	Cost
*DenseNet (Huang et al., 2017)	3.46	25.6	-
*HierEvol (Liu et al., 2018b)	3.75±0.12	15.7	300
*NAONet-WS (Luo et al., 2018)	3.53	3.1	0.4
*PNAS (Liu et al., 2018a)	3.41±0.09	3.2	225
*ENAS (Pham et al., 2018)	2.89	4.6	0.5
*NASNet-A (Zoph et al., 2018)	2.65	3.3	1800
*AmoebaNet-B (Real et al., 2019)	2.55±0.05	2.8	3150
*R-DARTS (Zela et al., 2020)	2.95±0.21	-	1.6
*GDAS (Dong and Yang, 2019)	2.93	3.4	0.2
*GTN (Such et al., 2019)	2.92±0.06	8.2	0.67
*SNAS (Xie et al., 2019)	2.85	2.8	1.5
*BayesNAS (Zhou et al., 2019)	2.81±0.04	3.4	0.2
*MergeNAS (Wang et al., 2020)	2.73±0.02	2.9	0.2
*NoisyDARTS (Chu et al., 2020b)	2.70±0.23	3.3	0.4
*ASAP (Noy et al., 2020)	2.68±0.11	2.5	0.2
*SDARTS (Chen and Hsieh, 2020)	2.61±0.02	3.3	1.3
*DropNAS (Hong et al., 2020)	2.58±0.14	4.1	0.6
*FairDARTS (Chu et al., 2019)	2.54	3.3	0.4
*DrNAS (Chen et al., 2020)	2.54±0.03	4.0	0.4
*DARTS-1st (Liu et al., 2019)	3.00±0.14	3.3	0.4
LPT-R18-DARTS-1st (ours)	**2.85**±0.09	2.7	0.6
*DARTS-2nd (Liu et al., 2019)	2.76±0.09	3.3	1.5
LPT-R18-DARTS-2nd (ours)	2.72±0.07	3.4	1.8
LPT-R50-DARTS-2nd (ours)	**2.68**±0.02	3.4	2.0
*DARTS$^-$ (Chu et al., 2020a)	2.59±0.08	3.5	0.4
†DARTS$^-$ (Chu et al., 2020a)	2.97±0.04	3.3	0.4
LPT-R18-DARTS$^-$ (ours)	2.74±0.07	3.4	0.6
$^\triangle$DARTS$^+$ (Liang et al., 2019a)	2.83±0.05	3.7	0.4
LPT-R18-DARTS$^+$ (ours)	**2.69**±0.05	3.6	0.5
*PC-DARTS (Xu et al., 2020)	**2.57**±0.07	3.6	0.1
LPT-R18-PC-DARTS (ours)	2.65±0.17	3.7	0.1
*P-DARTS (Chen et al., 2019)	2.50	3.4	0.3
LPT-R18-P-DARTS (ours)	2.58±0.14	3.3	0.5

outperforms LPT-R18-DARTS-2nd, where the former uses ResNet-50 as the data encoder in the tester while the latter uses ResNet-18. ResNet-50 has a better ability of learning representations than ResNet-18 since it is “deeper”: 50 layers versus 18 layers. This shows

Table 4: Results on ImageNet, including top-1 and top-5 classification errors on the test set, number of weight parameters (millions), and search cost (GPU days). * means the results are taken from DARTS$^-$ (Chu et al., 2020a) and DrNAS (Chen et al., 2020). The rest notations are the same as those in Table 2 in the main paper. The first row block shows networks designed by human manually. The second row block shows non-gradient based search methods. The third block shows gradient-based methods. ‡ means the results following the hyperparameters selected for CIFAR10/100. The hyperparameter for CIFAR100 is used when directly searching on ImageNet.

Method	Top-1 Error (%)	Top-5 Error (%)	Param (M)	Cost (GPU days)
*Inception-v1 (Szegedy et al., 2015)	30.2	10.1	6.6	-
*MobileNet (Howard et al., 2017)	29.4	10.5	4.2	-
*ShuffleNet 2× (v1) (Zhang et al., 2018)	26.4	10.2	5.4	-
*ShuffleNet 2× (v2) (Ma et al., 2018)	25.1	7.6	7.4	-
*NASNet-A (Zoph et al., 2018)	26.0	8.4	5.3	1800
*PNAS (Liu et al., 2018a)	25.8	8.1	5.1	225
*MnasNet-92 (Tan et al., 2019)	25.2	8.0	4.4	1667
*AmoebaNet-C (Real et al., 2019)	24.3	7.6	6.4	3150
*SNAS-CIFAR10 (Xie et al., 2019)	27.3	9.2	4.3	1.5
*BayesNAS-CIFAR10 (Zhou et al., 2019)	26.5	8.9	3.9	0.2
*PARSEC-CIFAR10 (Casale et al., 2019)	26.0	8.4	5.6	1.0
*GDAS-CIFAR10 (Dong and Yang, 2019)	26.0	8.5	5.3	0.2
*DSNAS-ImageNet (Hu et al., 2020)	25.7	8.1	-	-
*SDARTS-ADV-CIFAR10 (Chen and Hsieh, 2020)	25.2	7.8	5.4	1.3
*PC-DARTS-CIFAR10 (Xu et al., 2020)	25.1	7.8	5.3	0.1
*ProxylessNAS-ImageNet (Cai et al., 2019)	24.9	7.5	7.1	8.3
*FairDARTS-CIFAR10 (Chu et al., 2019)	24.9	7.5	4.8	0.4
*FairDARTS-ImageNet (Chu et al., 2019)	24.4	7.4	4.3	3.0
*DrNAS-ImageNet (Chen et al., 2020)	24.2	7.3	5.2	3.9
*DARTS$^+$-ImageNet (Liang et al., 2019a)	23.9	7.4	5.1	6.8
*DARTS$^-$-ImageNet (Chu et al., 2020a)	23.8	7.0	4.9	4.5
*DARTS$^+$-CIFAR100 (Liang et al., 2019a)	23.7	7.2	5.1	0.2
*DARTS-2nd-CIFAR10 (Liu et al., 2019)	26.7	8.7	4.7	1.5
LPT-R18-DARTS-2nd-CIFAR10 (ours)	25.3	7.9	4.7	1.8
*P-DARTS (CIFAR10) (Chen et al., 2019)	24.4	7.4	4.9	0.3
‡LPT-R18-P-DARTS-CIFAR10 (ours)	24.2	7.3	4.9	0.5
*P-DARTS (CIFAR100) (Chen et al., 2019)	24.7	7.5	5.1	0.3
‡LPT-R18-P-DARTS-CIFAR100 (ours)	24.0	7.1	5.3	0.5
*PC-DARTS-ImageNet (Xu et al., 2020)	24.2	7.3	5.3	3.8
‡LPT-R18-PC-DARTS-ImageNet (ours)	**23.4**	**6.8**	5.7	4.0

that a "stronger" tester can help the learner to learn better. With a more powerful data encoder, the tester can better understand examples in the test bank and can make better decisions in creating difficult and meaningful tests. Tests with better quality can evaluate the learner more effectively and help to improve the learner's learning capability. When our method is applied to PC-DARTS and P-DARTS, the performance difference resulting from ResNet-18 and ResNet-50 is not statistically significant. **Third**, our method LPT-R18-P-DARTS achieves the best performance among all methods, which further demonstrates the effectiveness of LPT in driving the frontiers of neural architecture search forward. **Fourth**, the number of weight parameters and search costs corresponding to our methods are on par with those in differentiable NAS baselines. This shows that LPT is able to search better-performing architectures without significantly increasing network size and search cost. A few additional remarks: 1) On CIFAR-100, DARTS-2nd with second-order approximation in the optimization algorithm is not advantageous compared with DARTS-1st which uses first-order approximation; 2) In our run of DARTS$^-$, we were not able to achieve the performance reported in (Chu et al., 2020a); 3) In our run of DARTS$^+$, in the architecture evaluation stage, we set the number of epochs to 600 instead of 2000 as in (Liang et al., 2019a), to ensure a fair comparison with other methods (where the epoch number is 600).

Table 3 shows the classification error (%), number of weight parameters (millions), and search cost (GPU days) of different NAS methods on CIFAR-10. As can be seen, applying our proposed LPT to DARTS-1st, DARTS-2nd, DARTS$^-$ (our run), and DARTS$^+$ significantly reduces the errors of these baselines. For example, with the usage of LPT, the error of DARTS-2nd is reduced from 2.76% to 2.68%. This further demonstrates the efficacy of our method in searching better-performing architectures, by creating tests with increasing levels of difficulty and improving the learner through taking these tests. On PC-DARTS and P-DARTS, applying our method does not yield better performance.

Table 4 shows the results on ImageNet, including top-1 and top-5 classification errors on the test set. In our proposed LPT-R18-PC-DARTS-ImageNet, the architecture is searched on ImageNet, where our method performs much better than PC-DARTS-ImageNet and achieves the lowest error (23.4% top-1 error and 6.8% top-5 error) among all methods in Table 4. In our methods including LPT-R18-P-DARTS-CIFAR100, LPT-R18-P-DARTS-CIFAR10, and LPT-R18-DARTS-2nd-CIFAR10, the architectures are searched on CIFAR-10 or CIFAR-100 and evaluated on ImageNet, where these methods outperform their corresponding baselines P-DARTS-CIFAR100, P-DARTS-CIFAR10, and DARTS-2nd-CIFAR10. These results further demonstrate the effectiveness of our method.

4.4. Ablation Studies

In order to evaluate the effectiveness of individual modules in LPT, we compare the full LPT framework with the following ablation settings.

- **Ablation setting 1**. In this setting, the tester creates tests solely by maximizing their level of difficulty, without considering their meaningfulness. Accordingly, the second stage in LPT where the tester learns to perform a target-task by leveraging the created tests is removed. The tester directly learns a selection scalar $s(d) \in [0,1]$ for each example d in the test bank without going through a data encoder or test creator. The corresponding formulation is:

$$\begin{array}{l} \max_S \min_A \quad \frac{1}{\sum_{d\in D_b} s(d)} \sum_{d\in D_b} s(d)\ell(A, W^*(A), d) \\ s.t.\ \ W^*(A) = \min_W \quad L\left(A, W, D_{ln}^{(\mathrm{tr})}\right) \end{array} \tag{17}$$

Table 5: Results for ablation setting 1. "Difficult only" denotes that the tester creates tests solely by maximizing their level of difficulty, without considering their meaningfulness, i.e., the tester does not use the tests for learning to perform the target task. "Difficult + meaningful" denotes the full LPT framework where the tester creates tests by maximizing both difficulty and meaningfulness.

Method	Error (%)
Difficult only (DARTS-2nd, CIFAR-100)	20.38±0.17
Difficult + meaningful (DARTS-2nd, CIFAR-100)	**19.47**±0.20
Difficult only (P-DARTS, CIFAR-100)	18.12±0.11
Difficult + meaningful (P-DARTS, CIFAR-100)	**16.28**±0.10
Difficult only (DARTS-2nd, CIFAR-10)	2.79±0.06
Difficult + meaningful (DARTS-2nd, CIFAR-10)	**2.72**±0.07

where $S = \{s(d)|d \in D_b\}$. In this study, λ and γ are both set to 1. The data encoder of the tester is ResNet-18. For CIFAR-100, LPT is applied to P-DARTS and DARTS-2nd. For CIFAR-10, LPT is applied to DARTS-2nd.

- **Ablation setting 2**. In this setting, in the second stage of LPT, the tester is trained solely based on the created test, without using the training data of the target task. The corresponding formulation is:

$$\begin{array}{l} \max_C \min_A \quad \frac{1}{|\sigma(C,E^*(C),D_b)|} L\left(A, W^*\left(A\right), \sigma\left(C, E^*(C), D_b\right)\right) \\ \qquad\qquad -\lambda L\left(E^*(C), X^*(C), D_{tt}^{(\text{val})}\right) \\ s.t.\;\; E^*(C), X^*(C) = \min_{E,X} \quad L\left(E, X, \sigma\left(C, E, D_b\right)\right) \\ \qquad W^*\left(A\right) = \min_W \quad L\left(A, W, D_{ln}^{(\text{tr})}\right) \end{array} \tag{18}$$

In this study, λ and γ are both set to 1. The data encoder of the tester is ResNet-18. For CIFAR-100, LPT is applied to P-DARTS and DARTS-2nd. For CIFAR-10, LPT is applied to DARTS-2nd.

- Ablation study on λ. We are interested in how the learner's performance varies as the tradeoff parameter λ in Eq.(3) increases. In this study, the other tradeoff parameter γ in Eq.(3) is set to 1. For both CIFAR-100 and CIFAR-10, we randomly sample 5K data from the 25K training and 25K validation data, and use it as a test set to report performance in this ablation study. The rest 45K data is used as before. Tester's data encoder is ResNe-18. LPT is applied to P-DARTS.

- Ablation study on γ. We investigate how the learner's performance varies as c increases. In this study, the other tradeoff parameter λ is set to 1. Similar to the ablation study on λ, on 5K randomly-sampled test data, we report performance of architectures searched under different values of γ. Tester's data encoder is ResNe-18. LPT is applied to P-DARTS.

Table 6: Results for ablation setting 2. "Test only" denotes that the tester is trained only using the created test to perform the target task. "Test + training" denotes that the tester is trained using both the test and the training data of the target task.

Method	Error (%)
Test only (DARTS-2nd, CIFAR-100)	19.81±0.06
Test + training (DARTS-2nd, CIFAR-100)	**19.47**±0.20
Test only (P-DARTS, CIFAR-100)	17.54±0.07
Test + training (P-DARTS, CIFAR-100)	**16.28**±0.10
Test only (DARTS-2nd, CIFAR-10)	2.75±0.03
Test + training (DARTS-2nd, CIFAR-10)	**2.72**±0.07

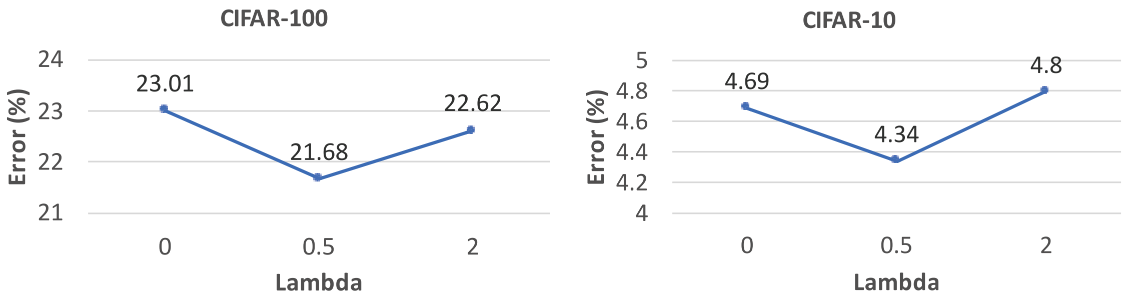

Figure 3: How errors change as λ increases.

Table 5 shows the results for ablation setting 1. As can be seen, on both CIFAR-10 and CIFAR-100, creating tests that are both difficult and meaningful is better than creating tests solely by maximizing difficulty. The reason is that a difficult test could be composed of bad-quality examples such as outliers and incorrectly-labeled examples. Even a highly-accurate model cannot achieve good performance on such erratic examples. To address this problem, it is necessary to make the created tests meaningful. LPT achieves meaningfulness of the tests by making the tester leverage the created tests to perform the target task. The results demonstrate that this is an effective way of improving meaningfulness.

Table 6 shows the results for ablation setting 2. As can be seen, for both CIFAR-100 and CIFAR-10, using both the created test and the training data of the target task to train the tester performs better than using the test only. By leveraging the training data, the data encoder can be better trained. And a better encoder can help to create higher-quality tests.

Figure 3 shows how classification errors change as λ increases. As can be seen, on both CIFAR-100 and CIFAR-10, when λ increases from 0 to 0.5, the error decreases. However, further increasing λ renders the error to increase. From the tester's perspective, λ explores a tradeoff between difficulty and meaningfulness of the tests. Increasing λ encourages the tester to create tests that are more meaningful. Tests with more meaningfulness can more reliably evaluate the learner. However, if λ is too large, the tests are biased to be more meaningful but less difficult. Lacking enough difficulty, the tests may not be compelling

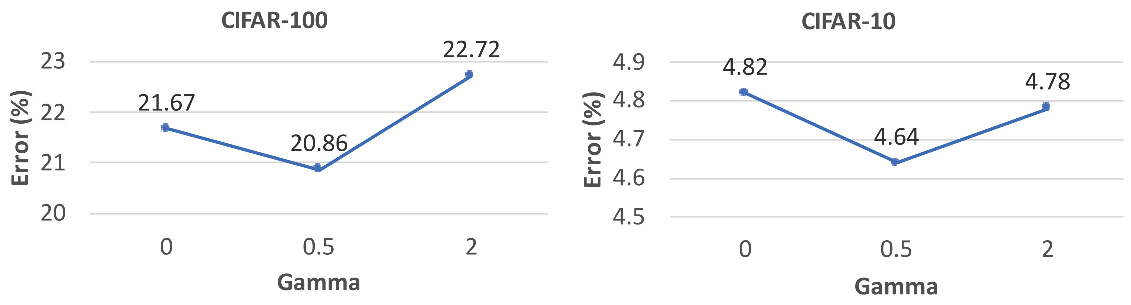

Figure 4: How errors change as γ increases.

enough to drive the learner for improvement. Such a tradeoff effect is observed in the results on CIFAR-10 as well.

Figure 4 shows how classification errors change as γ increases. As can be seen, on both CIFAR-100 and CIFAR-10, when γ increases from 0 to 0.5, the error decreases. However, further increasing γ renders the error to increase. Under a larger γ, the created test plays a larger role in training the tester to perform the target task. This implicitly encourages the test creator to generate tests that are more meaningful. However, if γ is too large, training is dominated by the created test which incurs the following risk: if the test is not meaningful, it will result in a poor-quality data-encoder which degrades the quality of created tests.

5. Conclusions

In this paper, we propose a new machine learning approach – learning by passing tests (LPT), inspired by the test-driven learning technique of humans. In LPT, a tester model creates a sequence of tests with growing levels of difficulty. A learner model continuously improves its learning ability by striving to pass these increasingly more-challenging tests. We propose a multi-level optimization framework to formalize LPT where the tester learns to select hard validation examples that render the learner to make large prediction errors and the learner refines its model to rectify these prediction errors. Our framework is applied for neural architecture search and achieves significant improvement on CIFAR-100, CIFAR-10, and ImageNet.

References

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*, 2019.

Francesco Paolo Casale, Jonathan Gordon, and Nicoló Fusi. Probabilistic neural architecture search. *CoRR*, abs/1902.05116, 2019.

Xiangning Chen and Cho-Jui Hsieh. Stabilizing differentiable architecture search via perturbation-based regularization. *CoRR*, abs/2002.05283, 2020.

Xiangning Chen, Ruochen Wang, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. Drnas: Dirichlet neural architecture search. *CoRR*, abs/2006.10355, 2020.

Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *ICCV*, 2019.

Xiangxiang Chu, Tianbao Zhou, Bo Zhang, and Jixiang Li. Fair DARTS: eliminating unfair advantages in differentiable architecture search. *CoRR*, abs/1911.12126, 2019.

Xiangxiang Chu, Xiaoxing Wang, Bo Zhang, Shun Lu, Xiaolin Wei, and Junchi Yan. DARTS-: robustly stepping out of performance collapse without indicators. *CoRR*, abs/2009.01027, 2020a.

Xiangxiang Chu, Bo Zhang, and Xudong Li. Noisy differentiable architecture search. *CoRR*, abs/2005.03566, 2020b.

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

Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four GPU hours. In *CVPR*, 2019.

Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *International Conference on Machine Learning*, pages 1180–1189, 2015.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014a.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014b.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016b.

Weijun Hong, Guilin Li, Weinan Zhang, Ruiming Tang, Yunhe Wang, Zhenguo Li, and Yong Yu. Dropnas: Grouped operation dropout for differentiable architecture search. In *IJCAI*, 2020.

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

Shoukang Hu, Sirui Xie, Hehui Zheng, Chunxiao Liu, Jianping Shi, Xunying Liu, and Dahua Lin. DSNAS: direct neural architecture search without parameter retraining. In *CVPR*, 2020.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

Lu Jiang, Deyu Meng, Shoou-I Yu, Zhenzhong Lan, Shiguang Shan, and Alexander Hauptmann. Self-paced learning with diversity. *Advances in Neural Information Processing Systems*, 27:2078–2086, 2014.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

M Pawan Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. In *Advances in neural information processing systems*, pages 1189–1197, 2010.

Hanwen Liang, Shifeng Zhang, Jiacheng Sun, Xingqiu He, Weiran Huang, Kechen Zhuang, and Zhenguo Li. DARTS+: improved differentiable architecture search with early stopping. *CoRR*, abs/1909.06035, 2019a.

Hanwen Liang, Shifeng Zhang, Jiacheng Sun, Xingqiu He, Weiran Huang, Kechen Zhuang, and Zhenguo Li. Darts+: Improved differentiable architecture search with early stopping. *arXiv preprint arXiv:1909.06035*, 2019b.

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *ECCV*, 2018a.

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. In *ICLR*, 2018b.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. In *ICLR*, 2019.

Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *NeurIPS*, 2018.

Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet V2: practical guidelines for efficient CNN architecture design. In *ECCV*, 2018.

Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher-student curriculum learning. *IEEE transactions on neural networks and learning systems*, 2019.

Asaf Noy, Niv Nayman, Tal Ridnik, Nadav Zamir, Sivan Doveh, Itamar Friedman, Raja Giryes, and Lihi Zelnik. ASAP: architecture search, anneal and prune. In *AISTATS*, 2020.

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *ICML*, 2018.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

Michelle Shu, Chenxi Liu, Weichao Qiu, and Alan Yuille. Identifying model weakness with adversarial examiner. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 11998–12006, 2020.

Felipe Petroski Such, Aditya Rawal, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data. *CoRR*, abs/1912.07768, 2019.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, 2019.

Xiaoxing Wang, Chao Xue, Junchi Yan, Xiaokang Yang, Yonggang Hu, and Kewei Sun. Mergenas: Merge operations into one for differentiable architecture search. In *IJCAI*, 2020.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: stochastic neural architecture search. In *ICLR*, 2019.

Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. PC-DARTS: partial channel connections for memory-efficient architecture search. In *ICLR*, 2020.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI*, 2017.

Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *ICLR*, 2020.

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, 2018.

Hongpeng Zhou, Minghao Yang, Jun Wang, and Wei Pan. Bayesnas: A bayesian approach for neural architecture search. In *ICML*, 2019.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018.