Automated learning rate scheduling with in-training information

Yongjun Kim, Youngjun Yu

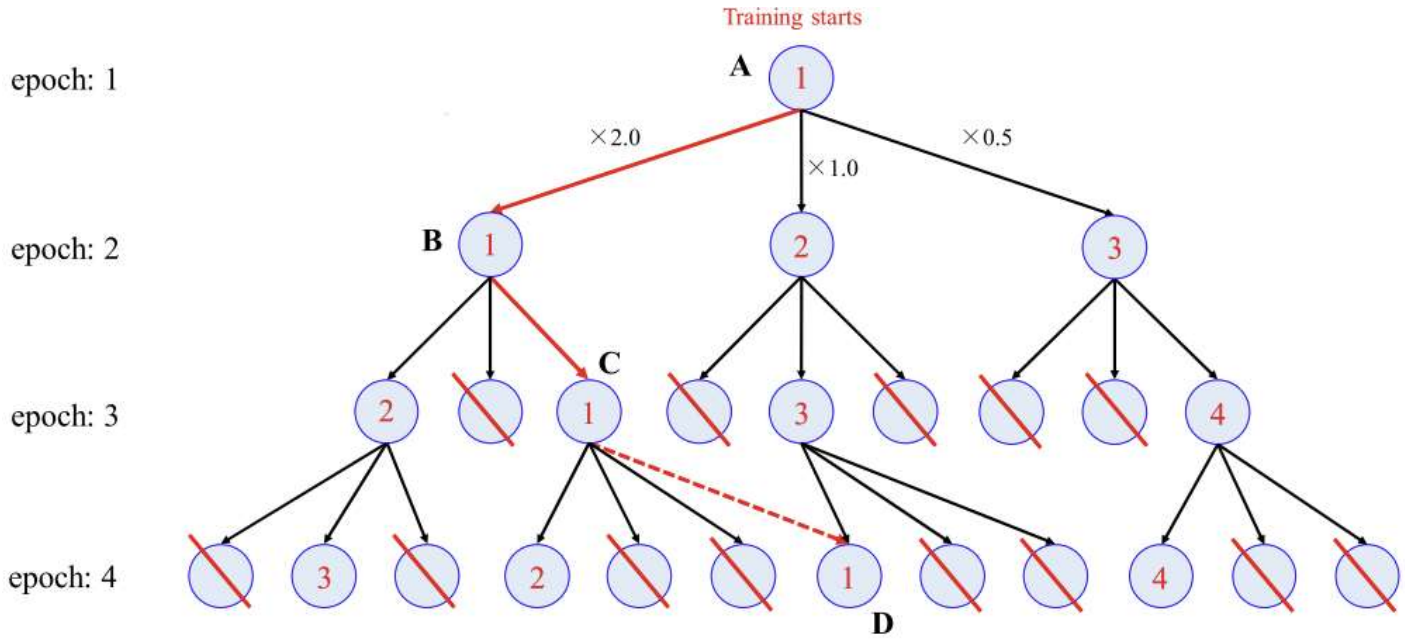
Table of Contents

Table of Contents

Motivation

- It is common to adjust learning rate in training procedure by using pre-determined schemes
- However, simply applying pre-determined strategy for learning rate may not sufficient; to accurately reflect in-training dynamics, consideration of various factors such as model architecture and dataset is necessary, which is not trivial
- Therefore, there needs an automatic and flexible method using in-training information which indirectly reflects those factors

Background: ALR technique (Adaptive Learning Rate Tree algorithm)



- ALR technique performs the training independently in parallel with several learning rates, choosing as the learning rate the one that has resulted in the smallest training loss.
- It uses breadth-first beam search to select an appropriate learning rate.
- The point is that several learning rates are stored in ascending order of training loss.

Background: ALR technique (Adaptive Learning Rate Tree algorithm)

Algorithm 1 ALR technique

Initialize: beam size: M;
number of branches: N;
scale factors: $\{r_n\}_{n=1}^{N}$;
learning rate: $\{\eta_m\}_{m=1}^{M}$;
weights and biases: θ, $\{\theta_m\}_{m=1}^{M}$;
number of batches for each epoch: B;

Input: training data: $\{x^{(b)}\}_{b=1}^{B}$, $\{y^{(b)}\}_{b=1}^{B}$;

$t \leftarrow 0$

repeat

  for $m = 1, 2, \cdots, M$ **do**

    for $n = 1, 2, \cdots, N$ **do**

      $\eta_{m,n} \leftarrow \eta_m \cdot r_n$

      $\theta_{m,n} \leftarrow \theta_m$

      for $b = 1, 2, \cdots, B$ **do**

        $E_{m,n} \leftarrow E(\theta_{m,n}; x^{(b)}; y^{(b)})$

        $\theta_{m,n} \leftarrow \theta_{m,n} - \eta_{m,n} \cdot \nabla_\theta E_{m,n}$

      end for

    end for

  end for

  $\theta \leftarrow \theta_{m_t,n_t}$ **where** $m_t, n_t \leftarrow \underset{m,n}{argmin}\{E\}$

  Sort $\{(\eta_{m,n}, \theta_{m,n}) \mid 1 \leqq m \leqq M, 1 \leqq n \leqq N\}$ in ascending order of $\{E_{m,n}\}$ and Save the first M pairs as $(\eta_1, \theta_1), \cdots, (\eta_M, \theta_M)$.

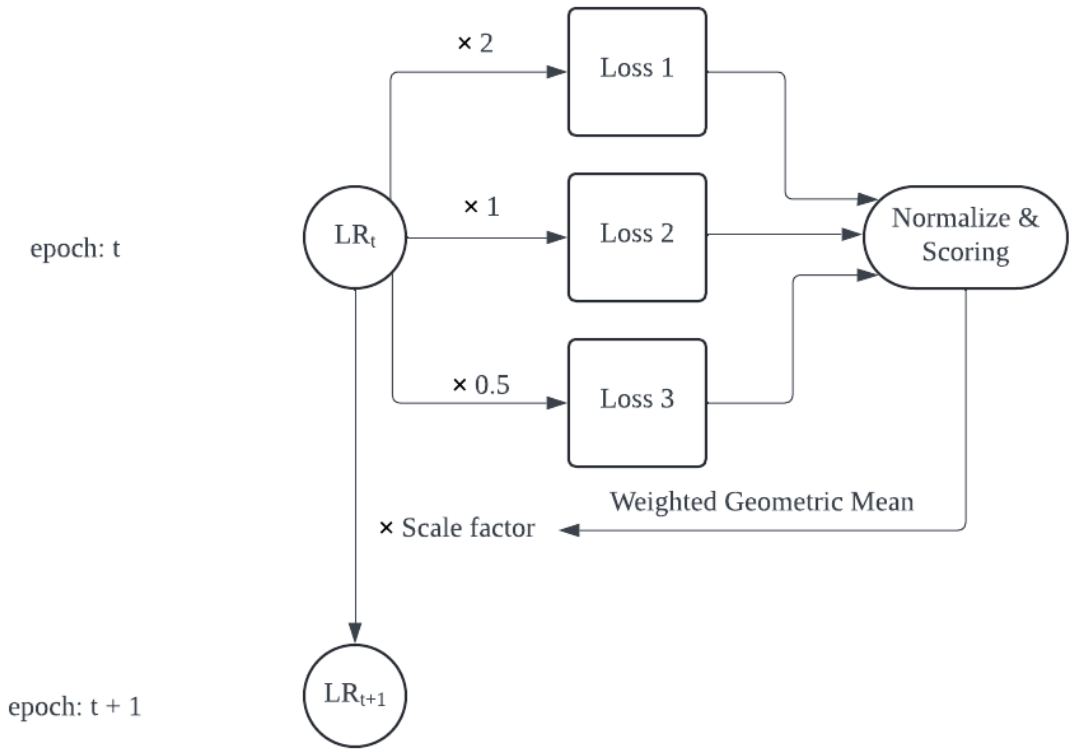
  $t \leftarrow t + 1$

until converged

- Training continues until it converges.
- It incurs a high computational cost because we need to calculate the training loss for various learning rates(beam size M) and different scale factors(number of branches N).
- For a single epoch, we need to calculate a total of M*N training losses and then select the smallest M from among them.

Table of Contents

Design & Methodology:



- We have overcome the problem of high computational cost with the ALR technique.
- We assigned scores based on the normalized training losses of learning rate values applied with various scale factors, and used these scores to calculate a weighted geometric mean.
- This weighted geometric mean is then used as the scale factor for the current epoch.

Design & Methodology:

Algorithm 1: Automated learning rate scheduler

Input: model: θ_t, factor list, learning rate: η_t, score function: *score*, loss function: L, Randomly selected batch: B
Output: η_{t+1}

for f *in* *factor list* **do**
    $\theta'_t = \text{copy}(\theta_t)$
    $\nabla_{\theta_t} L \leftarrow forward\&backprop$
    $\theta'_{t+1} = \theta'_t - f \cdot \eta_t \cdot \nabla_{\theta_t} L$
    Loss_list $\leftarrow L(\theta'_{t+1}, B)$

Normalized_Loss_list = Normalize(Loss_list)

for *loss* *in* *Normalized_Loss_list* **do**
    score list $\leftarrow$ *score*(*loss*)

η_{t+1} = weighted_geometry_average(factor list, score list) $\cdot \eta_t$
return

- We used exponential score function(e^{-x}), to amplify the influence of factors that results in significantly smaller losses.
- We use the geometric mean because it prevents the average from being skewed by extreme values, leading to more stable and gradual changes in the learning rate.
- ALR technique requires calculating the training loss for beam size (M) * number of factors (N), but this algorithm only needs to calculate N training losses, making it computationally less expensive.

Design & Methodology:

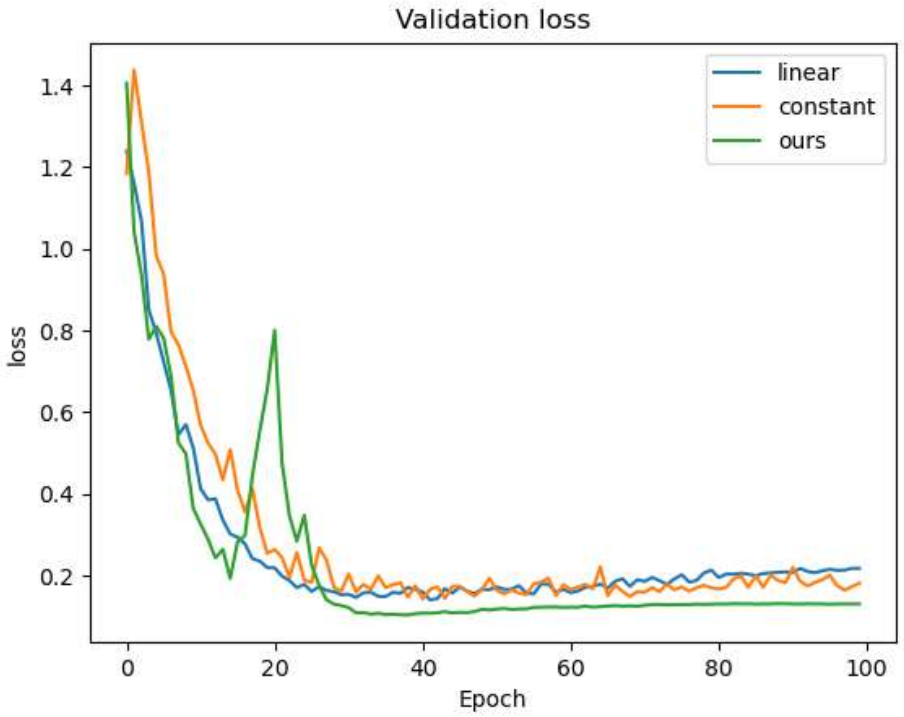
Setup

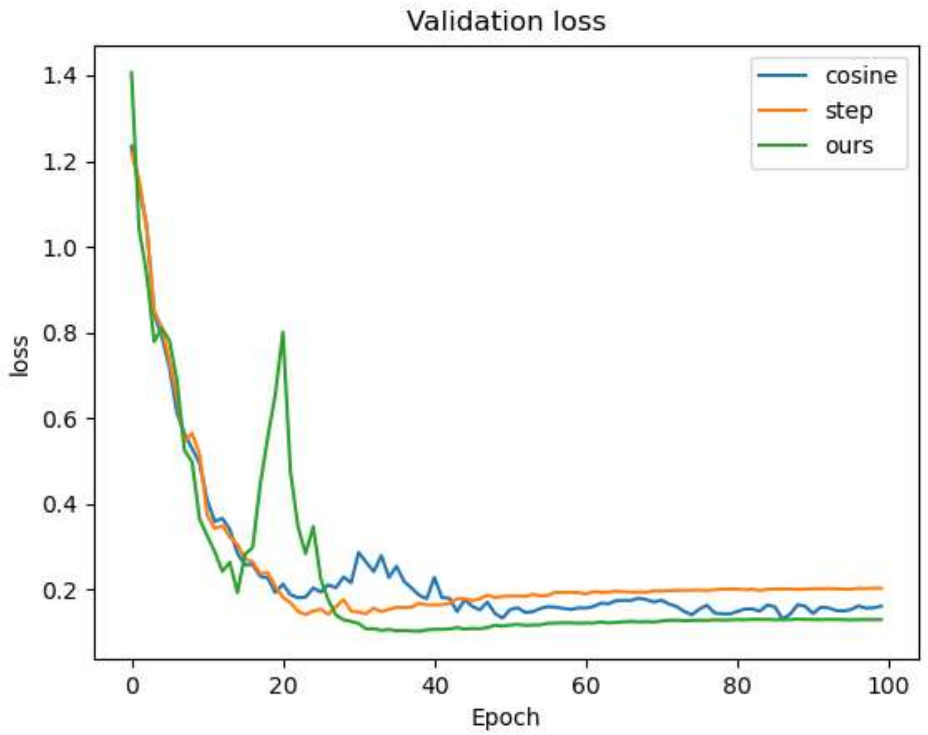
- We use ResNet18 model and CIFAR10 dataset unless otherwise specified.
- Hyperparameters of already existing schedulers are set to default value of pytorch except for initial learning rate.
- Initial learning rates are set to 0.005, and we conduct experiments 100 epochs, with 32 batch size
- Factors for our scheduler are consisted as follows
  - Factor_3 : [2, 1, 0.5]
  - Factor_5 : [4, 2, 1, 0.5, 0.125]
  - Factor_7 : [8, 4, 2, 1, 0.5, 0.25, 0.125]

Table of Contents

Results

Results: Performance of our scheduler comparing with exist methods





- Our method converges faster, and has better performance in terms of validation loss

Results

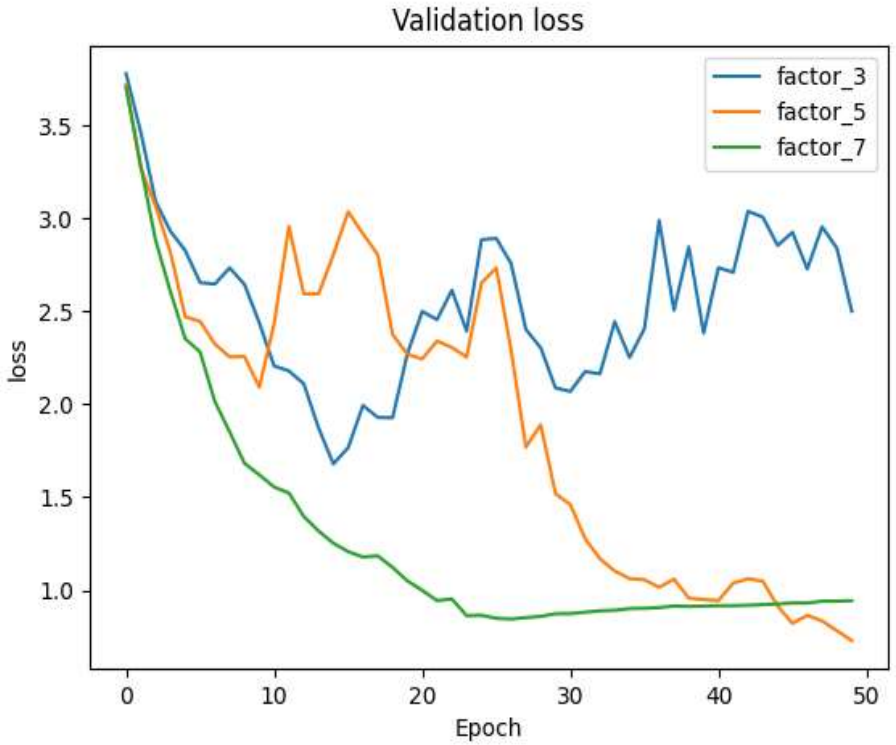
Results: Performance of our scheduler comparing with exist methods

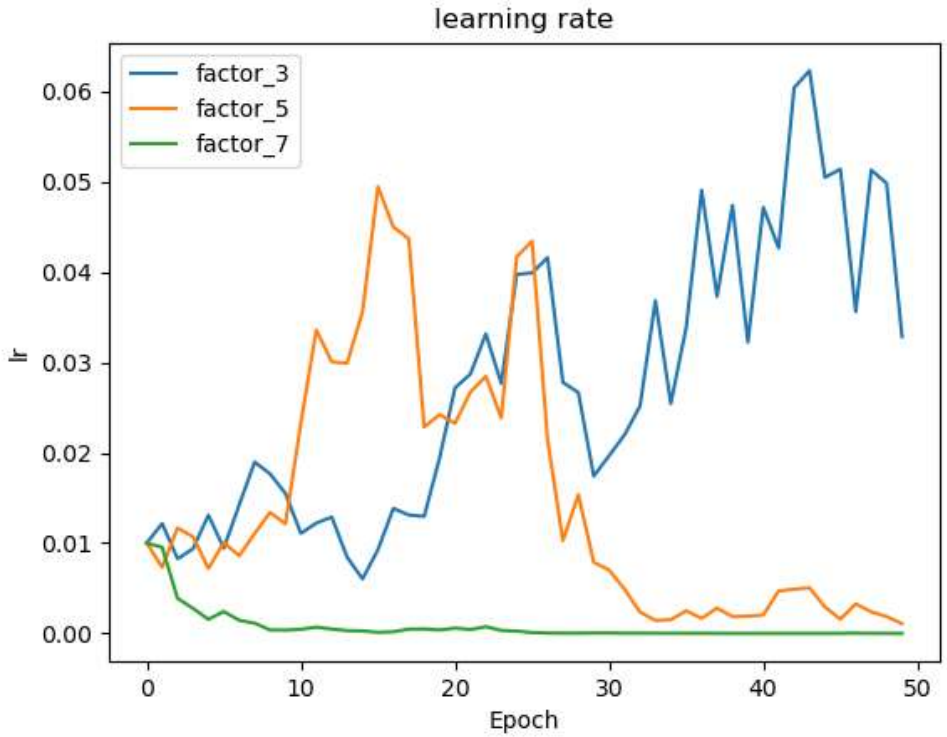
scheduler	CIFAR-10 test accuracy(%)
Constant	83.5
linear	85.2
cosine	85.0
step	84.4
ours	**87.0**

- Our method shows better test accuracy on CIFAR-10 dataset

Results

Results: Effects of # of factors (cifar100)

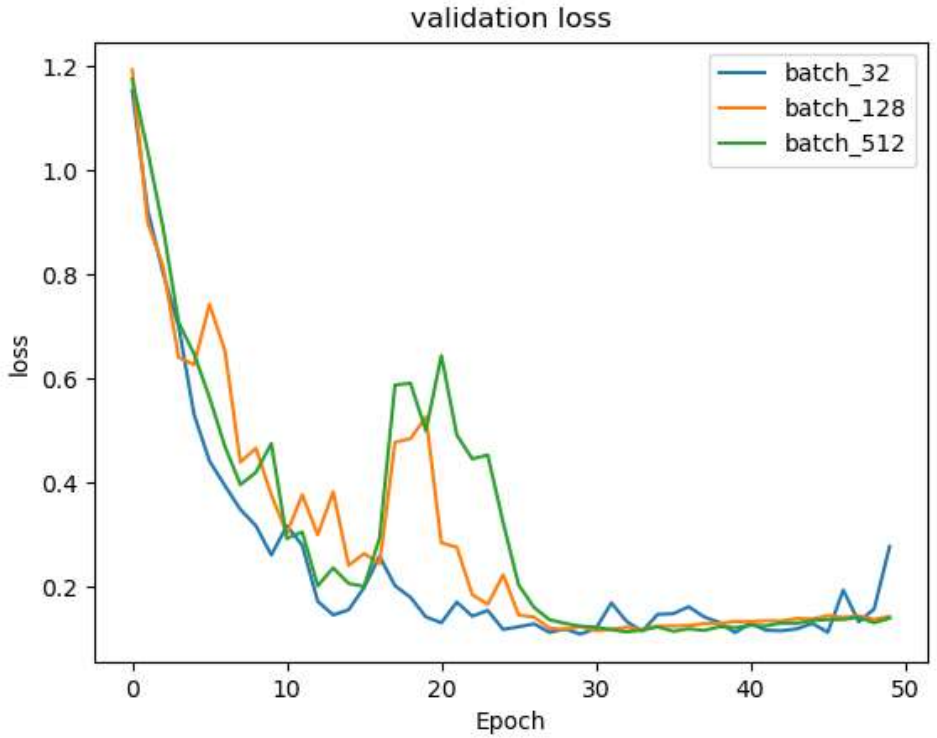




- Increasing the number of factors does not always increase the performance, but it can make the model stuck in suboptimal local minimum.

Results

Results: Effects of # of batch sizes used for factor scoring



- Smaller batch sizes, particularly 32, provide more stable and lower validation loss, suggesting better generalization performance for our algorithm.

Results

Can we use arbitrary learning rate without any consideration if we use this scheduler?

- As we set initial learning rate as 1, which is very large, with factors of [8, 4, 2, 1, 0.5, 0.25, 0.125], it diverges.
- Prior knowledge about learning rate is still needed even if we use this scheduler
- If the range of factors increases, it may find appropriate learning rate without divergence

Limitation & Future work

1. More GPU memory is needed to use this scheduler
   - Model should be copied to calculate loss value

2. It needs additional computation cost unlike other predetermined learning rate scheduler, and it is not very trivial.

3. We used an exponential score function(e^{-x}), but this function does not adequately account for the normalized training loss values. Therefore, a more appropriate score function design is needed to better reflect the impact of these values.